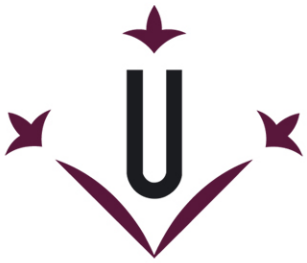
Tesis Doctoral

Infraestructura Software de Soporte al Desarrollo de Interfaces de Usuario Plásticas bajo una Visión Dicotómica

Escola Politècnica Superior

Universitat de Lleida



Lleida, septiembre de 2007

Tesis Doctoral desarrollada por *Montserrat Sendín* y dirigida por el Dr. *Jesús Lorés Vidal* y, posteriormente por el Dr. *César A. Collazos Ordónez* y el Dr. *Víctor Manuel López-Jaquero* para optar al título de Doctor por la Universidad de Lleida, especialidad Informática, por el Doctorado en Ingeniería Rural y Agroforestal de la Universitat de Lleida (España)

Prefacio

Esta Tesis se presenta como parte de los requisitos para optar al grado académico de *Doctor por la Universidad de Lleida, especialidad Informática* por el programa de Doctorado en Ingeniería Rural y Agroforestal de la *Universitat de Lleida* (España). No ha sido presentada previamente para la obtención de otro título en esta Universidad u otras. La misma contiene los resultados obtenidos en investigaciones llevadas a cabo en el *Grup de Recerca en Interacció Home–Ordinador i Bases de Dades GRIHO* de la Escola Politécnica Superior de la Universitat de Lleida. Se han trabajado las áreas de la Plasticidad de las Interfaces de Usuario y el enfoque Basado en Modelos propias de la Interacción Persona–Ordenador.

La Tesis doctoral que aquí se presenta ha sido desarrollada bajo la dirección del Dr. Jesús Lorés Vidal[**] y, posteriormente, del Dr. César A. Collazos Ordónez y del Dr. Víctor Manuel López-Jaquero.

Montserrat Sendín

Lleida, septiembre de 2007

Escola Politécnica Superior

Universitat de Lleida

[**] In memoriam.

A mi familia.

Agradecimientos

En primer lugar, me gustaría agradecer a mi familia su paciencia y apoyo desinteresado, que en estos últimos meses se ha convertido en una verdadera prueba de aplomo, incluso para mis padres. Quisiera mencionar a aquellos que lo han vivido día a día: mis hijos, Gerard y Sergi, que han sabido aceptar cuál era mi prioridad en la fase de redacción, y que han colaborado en la parte anímica con sus entretenidas conversaciones y sus actuaciones cómicas en los escasos espacios de tiempo compartidos; y, por supuesto, Jordi, pues sin su apoyo, comprensión y colaboración, estas líneas hubieran tenido que esperar quién sabe cuánto tiempo más en ser escritas.

A continuación debo mencionar a todos mis compañeros de área y a los compañeros del grupo de investigación GRIHO, que han tolerado mi ausencia y encapsulamiento durante esta última etapa, favoreciéndome en el reparto de la docencia y animándome en todo momento.

A Juan Manuel Gimeno, mi compañero de despacho, que siempre está a punto para compartir sus conocimientos, y que una vez te ha resuelto el problema responde con su típico: *"A disposar"* (*"a disponer"* en castellano).

A Mª Paula González, compañera de GRIHO con la que he compartido muchas vivencias y situaciones, y a la que agradezco el haberme contagiado su pasión por el Látex, facilitándome el material y soporte suficiente como para disuadir cualquier excusa para no utilizarlo en la preparación de este documento. También me gustaría mencionar a su esposo, Carlos Chesnévar, pues me supo aconsejar en momentos puntuales, transmitiendo su experiencia como investigador.

A Jesús Lorés le agradezco su obra, a la que dedicó tanto, de la que me beneficio por partida doble al ser miembro tanto de GRIHO como de AIPO, y que con su entusiasmo y entrega fue estableciendo una comunidad en la que hoy muchos podemos decir que nos sentimos como en una gran familia.

A mis actuales directores, cuya participación ha sido clave en momentos cruciales.

De César Collazos destaco que creyera fehacientemente en mi trabajo, dándome el empujón moral y transmitiéndome el talante propio de un investigador, ávido por ampliar horizontes.

A Víctor López le agradezco la confianza depositada en mí al ofrecerse para colaborar activamente en la co-dirección de este trabajo. A pesar de la brevedad del periodo, la

intensidad en el intercambio de impresiones, la fluidez de la comunicación y sus numerosas aportaciones bien han suplido el periodo previo a su incorporación.

Durante los últimos meses hay momentos en los que saber que no estás solo sirve de mucho. Yo me he sentido acompañada y apoyada, a pesar de tener a mis co-directores a muchos kilómetros de distancia. Siempre he recibido de ellos la sugerencia, la alternativa y el apoyo necesario para superar cada duda o dificultad.

A incontables compañeros de AIPO, que a lo largo de estos años se han interesado por mi progreso, me han animado a dar nuevos pasos y han contribuido de una u otra forma.

No debo descuidar la labor de los *referees*, en especial Juan Manuel Murillo, que a pesar de no ser de AIPO se ofreció desinteresadamente a revisar esta tesis como experto en las técnicas de Ingeniería del Software aplicadas.

No quisiera dejar de nombrar a Jordi Viladrich, un proyectista modelo, por su interés en el tema, su saber hacer y disciplina de trabajo. Demostró confianza en mí, creyó en mi propuesta y supo ilusionarse en el trabajo que desempeñó como parte de esta tesis.

Y, para concluir, mil disculpas a todo aquel que, a pesar de haberme ayudado no he recordado en el momento de redactar estas líneas. Les pediría que aceptaran mi más sincero agradecimiento.

A todos vosotros, gracias por confiar en mí.

Resumen

En una sociedad claramente influenciada por las nuevas tecnologías, los nuevos avances en computación móvil han supuesto un cambio trepidante en los hábitos de interacción con los sistemas y de acceso a una información omnipresente. Cualquier entorno puede devenir un escenario potencial para la realización de tareas interactivas, inclusive en colaboración con otros usuarios, sin renunciar a la movilidad. Paralelamente, las aplicaciones tienden a ofrecer un soporte colaborativo, ya sea por motivos sociales o de productividad. Hoy en día la ubicuidad de la interacción es un hecho plausible.

Este escenario introduce la necesidad de aportar soluciones software que incluyan el factor de diversidad en el contexto de uso. Aunque las técnicas desarrolladas hasta ahora en la disciplina de la Interacción Persona Ordenador proporcionan una base sólida, este tipo de aspectos no han sido cubiertos de manera plenamente satisfactoria. Se requieren soportes flexibles, exhaustivos y sistemáticos para el diseño y ejecución de interfaces de usuario capaces de adaptarse a las diversas situaciones que pueden surgir durante el proceso de interacción. Precisamente esa flexibilidad para soportar variaciones de distinta índole en el contexto de uso sin descuidar la usabilidad define el concepto de *plasticidad de la interfaz de usuario.*

En el estudio de esta problemática se perfilan dos problemas diferenciados: (a) el incremento de la complejidad en el diseño de este tipo de interfaces; y (b) una creciente demanda de adaptación dinámica y evolutiva. En este trabajo se propone el estudio por separado de estos dos retos, caracterizados a través de dos sub-conceptos de plasticidad denominados respectivamente *plasticidad explícita* y *plasticidad implícita*. El enfoque basado en esta división, el cual delimita el espacio de solución del problema, recibe el nombre de *"Visión Dicotómica de Plasticidad"*, y es el que fundamenta este trabajo.

La *visión dicotómica* combina dos motores de naturaleza distinta que se enmarcan en los lados opuestos de una arquitectura cliente-servidor. Siguiendo la misma terminología, estos motores se denominan respectivamente *Motor de Plasticidad Explícita*, que ofrece soporte en el diseño y desarrollo de interfaces de usuario plásticas y sensibles al grupo, así como para el mantenimiento de un *conocimiento compartido*; y *Motor de Plasticidad Implícita*, capaz de detectar el entorno y reaccionar adecuadamente ante un conjunto preestablecido de circunstancias, entre las que se incluyen las relativas al trabajo en grupo. La incorporación de las condiciones de grupo en la caracterización del contexto de uso y en el proceso de plasticidad resulta innovadora y abre un camino de solución conjunta para la plasticidad y los problemas intrínsecos a los escenarios colaborativos.

Resum

En una societat clarament influenciada per les noves tecnologies, els nous avanços en computació mòbil han suposat un canvi trepidant en els hàbits d'interacció amb els sistemes i d'accés a una informació omnipresent. Qualsevol entorn pot devenir un escenari potencial per a la realització de tasques interactives, fins i tot en col·laboració amb altres usuaris, sense renunciar a la mobilitat. Paral·lelament, les aplicacions tendeixen a oferir un suport col·laboratiu, ja sigui per motius socials o de productivitat. Avui en dia la ubiqüitat de la interacció és un fet plausible.

Aquest escenari introdueix la necessitat d'aportar solucions software que incloguin el factor de diversitat en el context d'ús. Encara que les tècniques desenvolupades fins ara en la disciplina de la Interacció Persona Ordinador proporcionen una base sòlida, aquest tipus d'aspectes no han estat coberts de manera plenament satisfactòria. Es requereixen suports flexibles, exhaustiu i sistemàtics per al disseny i execució d'interfases d'usuari capaces d'adaptar-se a les diverses situacions que poden sorgir durant el procés d'interacció. Precisament aquesta flexibilitat per suportar variacions de diferent índole en el context d'ús sense descuidar la usabilitat defineix el concepte de *plasticitat de la interfaç d'usuari*.

En l'estudi d'aquesta problemàtica es perfilen dos problemes diferenciats: (a) l'increment de la complexitat en el disseny d'aquest tipus d'interfases; i (b) una creixent demanda d'adaptació dinàmica i evolutiva. En aquest treball es proposa l'estudi per separat d'aquests dos reptes, caracteritzats a través de dos sub-conceptes de plasticitat denominats respectivament *plasticitat explícita* i *plasticitat implícita*. L'enfocament basat en aquesta divisió, el qual delimita l'espai de solució del problema, rep el nom de *"Visió Dicotòmica de Plasticitat"*, i és el que fonamenta aquest treball.

La *visió dicotòmica* combina dos motors de natura diferent que s'emmarquen en els dos costats oposats d'una arquitectura client-servidor. Seguint la mateixa terminologia, aquests motors s'anomenen respectivament *Motor de Plasticitat Explícita*, que ofereix suport en el disseny i desenvolupament d'interfases d'usuari plàstiques i sensibles al grup, així com per al manteniment d'un coneixement compartit; i *Motor de Plasticitat Implícita*, capaç de detectar l'entorn i reaccionar adequadament davant un conjunt preestablert de circumstàncies, entre les que s'inclouen les relatives al treball en grup. La incorporació de les circumstàncies de grup en la caracterització del context d'ús i en el procés de plasticitat resulta innovadora i obre un camí de solució conjunta per la plasticitat i els problemes intrínsecs als escenaris col·laboratius.

Abstract

In a society clearly influenced by new technologies, new advances in mobile computing have supposed a tremendous change in the habits of interaction with systems and access to an information that is undergoing omnipresent. For one thing, any environment can become a potential scenario for carrying out interactive tasks, even in collaboration with other users, and without renouncing mobility. Furthermore, applications tend to offer collaborative features, either for social or performance reasons. Today, ubiquity in interaction is a laudable fact.

This scenario introduces the need for contributing software solutions that tackle the diversity factor in the context of use. Although the techniques developed up to now in the Human-Computer Interaction discipline provide a solid foundation, these aspects have not been covered in an altogether satisfactory way. A flexible, comprehensive and systematic support for the design and execution of user interfaces able to adapt themselves to the situations likely to arise during interaction is required. It is precisely this flexibility to support different types of variations in the context of use without neglecting usability that defines the *plasticity of the user interface* concept.

In the study of this issue, two well-defined problems appear: (a) the increasing complexity in the design of these kinds of interfaces; and (b) an increasing demand for dynamic and evolving adaptation. This work proposes the separate study of these two challenges, which are characterized by means of two sub-concepts of plasticity called *explicit plasticity* and *implicit plasticity* respectively. The approach based on this division, which delimits the problem solution space, receives the name of *"Dichotomic View of Plasticity"*, and it lays the foundations of this work.

The *dichotomic view* combines two different types of engines that are located on the opposite sides of a client-server architecture. Following the same terminology, these engines are called respectively *Explicit Plasticity Engine*, which offers support to both the design and development of plastic and group-aware user interfaces and the maintenance of *shared-knowledge*; and *Implicit Plasticity Engine*, which is able to detect the environment and react appropriately under a pre-established set of circumstances, among them those related to the work in group. The inclusion of the group conditions in the context of use and the plasticity process turns out to be innovative and it opens a way of dealing with plasticity and the problems intrinsic to collaborative scenarios in a combined manner.

Índice general

Índice de figuras

Indice de Tablas

Parte I

Contextualización

Capítulo 1

Introducción

"In the 21st century the technology revolution will move into the everyday, the small and the invisible..."

Mark Weiser."The Computer for the 21st Century"

Este capítulo introductorio está dedicado a ofrecer una idea general acerca de la problemática objeto de estudio en esta tesis, a partir de la cual se identifican dos retos diferenciados dentro del campo, los cuales van a encaminar el estudio y desarrollo llevado a cabo en este trabajo, así como también la elaboración del presente documento.

Una vez presentada una primera aproximación del contexto actual del problema se formulan las hipótesis de partida, se relacionan los objetivos planteados inicialmente, y se clasifica la tesis, tratando de situar el trabajo a caballo entre las distintas disciplinas en las que se enmarca. Para finalizar se describe la estructura del documento tratando de sintetizar cuál es el propósito de cada unidad temática.

A lo largo de este documento aparecen diversas definiciones, algunas de ellas obtenidas de una cierta fuente bibliográfica y otras concebidas por el propio autor de esta tesis. El estilo empleado es distinto en cada caso, aspecto que se cree conveniente aclarar aquí.

En las definiciones obtenidas de la literatura aparece el concepto definido junto con el nombre del primer autor y el año entre paréntensis. La propia definición aparece a continuación en cursiva. En este caso las definiciones no se enumeran, a no ser que se muestren varias definiciones distintas acerca del mismo término. En las definiciones propias no aparece ninguna referencia a una fuente bibliográfica. El concepto definido aparece entre paréntesis y precedido por la cabecera "*Definición*", apropiadamente enumerada en relación al capítulo en que se encuentra. La definición va seguida de dos puntos, en este caso sin cursiva. Las heurísticas también aparecen enumeradas en relación al capítulo, siguiendo siempre el estilo utilizado en el primer tipo de definiciones.

1.1. Problemática y Motivación

Este documento comienza con una descripción de la problemática existente para flexibilizar el desarrollo de sistemas interactivos capaces de adaptar su operatividad, apariencia y, por supuesto, la manera en que el usuario se comunica con el ordenador (su interfaz) a los distintos entornos, plataformas de computación y potencial que nos brinda hoy en día la tecnología. A continuación se expone la panorámica actual y se analiza la implicación que este desarrollo tiene en la disciplina de la *Interacción Persona-Ordenador*. Finalmente se identifican dos retos concretos relacionados con la problemática objeto de estudio de la presente tesis: la *plasticidad de las interfaces de usuario*.

1.1.1. Panorámica actual

Esta sección presenta la panorámica actual con respecto al desarrollo tecnológico que está experimentando el sector de la *Computación Móvil*.

1.1.1.1. Empuje tecnológico

No cabe duda de que la nueva era tecnológica está produciendo cambios profundos en el modo de proceder de nuestras actividades diarias. La tecnología de la información está avanzando a un ritmo vertiginoso, tomando cada vez más terreno, acercándose a nuevos y más amplios sectores de usuarios y facilitando mayor número de tareas en un mayor número de dominios de aplicación. Sin duda, la explosión de Internet a principios de los 90[1] rompió barreras tanto en el sector empresarial como en la sociedad, que pasa a ser una *Sociedad de la Información*[2] [Del04]. Con la penetración de Internet, la información y el acceso a todo tipo de servicios deja de estar al alcance de sólo un sector privilegiado, alcanzando diversos ámbitos de la sociedad. Sin embargo, gran cantidad de limitaciones de tipo tecnológico (cableado, conectividad, movilidad, etc.) estaban pendientes de resolver.

Con la llegada de las comunicaciones inalámbricas, la mejora de la conectividad, el aumento del ancho de banda, el desarrollo de redes híbridas[3], la miniaturización, la proliferación de dispositivos cada vez más compactos, y el progreso de las tecnologías informáticas, entre otros avances, se ha modificado el ecosistema social y económico. Actualmente, el acceso a Internet supera ya cualquier "barrera arquitectónica", hasta hoy impuesta por el uso de cables o la necesidad de permanecer en un punto prefijado. La

[1] Fecha a la que se atribuye la creación de la página Web por Tim Berners-Lee.

[2] Hoy en día Internet alberga unos 51 millones de sitios y se calculan unos 800 millones de internautas.

[3] Capaces de utilizar todas las tecnologías de comunicaciones actuales, escogiendo la más adecuada a cada momento, y capaz de pasar de una a otra de manera invisible para el usuario.

tecnología permite a los usuarios estar en movimiento sin renunciar a la interacción ni a los recursos de red, que continúan estando al alcance de la mano con cada vez menos restricciones. Como consecuencia de este avance, las facilidades de acceder a todo tipo de información y servicios son cada vez mayores, posibilitando la computación y la realización de tareas diversas en múltiples y variados entornos y condiciones. Otro aspecto destacable es que este avance ha acercado la interacción a sectores de la población hasta ahora ajenos a la tecnología, al mismo tiempo que ha potenciado aquellos que ya disfrutaban de ella.

Así pues, la concepción del acceso a la información como actividad estática, a desarrollar en un entorno de escritorio, queda relegada a la historia, abriéndose el camino hacia nuevos paradigmas de interacción, inspirados en el acceso constante a la información (*sensibilidad al contexto, interfaces naturales, computación móvil, computación ubicua*, *computación pervasiva*, *inteligencia ambiental*, *everyday computing*, *interfaces físicas*, *interfaces emocionales*, *realidad aumentada,...*) [AM00]. Todos ellos -cada uno con su peculiaridad propia- persiguen trasladar la interacción al mundo real, donde los entornos que nos encontramos son múltiples y dinámicos, y donde la frontera entre lo cotidiano y lo computacional se difumina, tal y como imaginaba Mark Weiser (Xerox PARC) [Wei91].

> *"... Con la difusión de la informática, el espacio y los objetos de la vida cotidiana se convierten en entidades de interacción..." [Wei93]*

Weiser ya predijo a principios de los 90 que la edad de la *Computación Ubicua*[4] estaba al alcance de la mano [Wei93]. Ciertamente, nos encontramos ante el nacimiento de una era donde las interfaces de usuario (en adelante, IUs) están rompiendo su antiguo cascarón en el que han estado prisioneras durante los últimos 20 años [AAC+02].

1.1.1.2. Factores determinantes en el despunte de nuevos paradigmas

Existen diversos factores que están contribuyendo a la materialización de nuevos e innovadores paradigmas de interacción. A continuación se relacionan algunos de ellos.

- La proliferación de dispositivos y servicios de telefonía móvil está jugando un papel esencial.
  - Los teléfonos celulares se están fusionando con los paginadores digitales, las PDAs (Personal Digital Assistant) y los organizadores personales que, dadas

[4] Cualquier tecnología de computación que permite la interacción más allá de una simple estación de trabajo, incluyendo la tecnología de todo tipo de dispositivo portable (handheld), las pantallas interactivas de gran escala, la infraestructura de red inalámbrica y la tecnología de voz y visión por computador [DFAB03].

sus crecientes capacidades de computación, comunicación e interacción[5], componen auténticos ordenadores de bolsillo provistos de comunicación. Las PDAs incluyen, progresivamente, los servicios de telefonía, y a la inversa, los fabricantes hacen de los teléfonos móviles una verdadera PDA.
Las posibilidades que ofrecen son realmente amplias y variadas. Podríamos resumirlas en la capacidad de acceder a una gran variedad de contenidos, formatos y servicios.

- Las novedades en los servicios ofrecidos por este tipo de dispositivos tampoco parecen tener límite. Las operadoras contribuyen a este crecimiento. Un ejemplo lo constituye el uso del móvil como tarjeta monedero, o como mecanismo para el pago de billetes (Empresa Malagueña de Transportes[6]) a través de la plataforma Mobipay[7]. Otro ejemplo es la vinculación de la telefonía móvil con empresas de seguridad e instalación de alarmas, conectando la alarma de casa con el móvil a través de GPRS[8] (General Packet Radio Service).
  Otra muestra de ello es el hecho de que el teléfono móvil cada vez integra más funcionalidades, convirtiéndose en lo que a veces se denomina un "todo en uno" [Gal07]. Además de las existentes en la actualidad, próximamente ofrecerán, además, el servicio de televisión de bolsillo, la realización de videollamadas en tiempo real, así como multitud de servicios cada vez más adaptados a los clientes [Bur07]. En estos casos en que confluyen gran cantidad de servicios se suele hablar de *convergencia de prestaciones y de medios*[9].
- Si a todo ello se suma que los terminales son cada vez más compactos, livianos y potentes en la capacidad de proceso y almacenamiento[10], el progreso de la interacción en este tipo de dispositivos está garantizado. De hecho, cada vez hay más desarrolladores creando programas para móviles, y lo que es más significativo, las empresas ya los emplean en procesos críticos.

- Proliferación de otros tipos de dispositivos de computación, que se diversifican por la forma y finalidad.

[5] Mayor tamaño de pantalla, botones de navegación más especializados, teclados más completos e interfaces más agradables.
[6] Empresa Malagueña de Transportes EMT SAM. http://www.emtsam.es/
[7] Mobipay España, 2002. http://www.mobipay.es/
[8] Conexión inalámbrica continua a redes de datos basada en una tecnología de conmutación por paquetes en la que la información se transmite en pequeñas ráfagas de datos a través de una red basada en IP (Internet Protocol) [Cab07].
[9] Revista Latinoamericana de Comunicación Chasqui. http://www.comunica.org/chasqui/81/salaverria81.htm
[10] Prueba de ello es la aparición de móviles provistos de discos duro de más de 1 Gb.

  - Otros dispositivos de interacción que también contribuyen a un entorno ubicuo son: tabletPCs, televisores provistos de Internet (WebTV), iMacs[11], kioscos interactivos, pizarras electrónicas, pantallas murales, paneles interactivos provistos de acceso a Internet, los nuevos híbridos entre PDA, móvil y tabletPC y, por supuesto, los clásicos ordenadores personales.
    Además, la continua reducción del coste del hardware sin duda permite que sigan surgiendo nuevas oportunidades computacionales.

- Las posibilidades de conexión incrementan paulatinamente gracias, entre otras cosas, a la implantación de la tecnología UMTS[12] (Universal Mobile Telecommunication Services).

  - Este hecho supone una ocasión única de crear un mercado masivo para acercar a la sociedad de la información servicios móviles personalizados. Los datos apuntan a que en el año 2008 habrá un millón de teléfonos UMTS en España.
  - Aunque UMTS todavía no se ha establecido en el mercado, ya ha encontrado sucesora. Se trata de la tecnología HSDPA (High Speed Downlink Packet Access)[13], que promete velocidades de hasta 10 Mbps, es decir, la misma que ADSL[14], hablándose ya de una cuarta generación (4G).

- La aceptación por parte de la población en el uso de este tipo de dispositivos resulta crucial.

  - No hay duda de que el teléfono móvil de segunda y tercera generación (2G, 2,5G y 3G) se ha convertido en un producto de uso masivo. Por ejemplo, en España se logró ya en el 2005 una penetración superior al 90 % de la población [Gon05b]. Nunca antes un dispositivo tecnológico había conseguido un nivel tan alto de implantación en un espacio tan corto de tiempo.

Todo ello contribuye a reconocer que la visión de Weiser [Wei91] empieza a verse materializada, dando paso asimismo a la emergencia de otros campos más especializados, mencionados anteriormente. Un ejemplo es la aparición del concepto de *Computación Móvil* [IEE07]. Se trata de una nueva área de la tecnología que se centra en ofrecer soluciones a problemas que conllevan movilidad y que en la mayoría de los casos implican el consumo de información o de servicios Web ofrecidos por uno o varios servidores. Lo que distingue la

[11] Es el nombre de la línea de ordenadores de Apple Computer dirigida al mercado doméstico. Estos ordenadores tienen en común la integración del monitor con la CPU y un diseño innovador [Wik07].
[12] Caracterizada por soportar velocidades de transmisión de datos de hasta 8-10 Mbps.
[13] Facilita la descarga de datos.
[14] Asymmetric Digital Subscriber Line

Computación Móvil de la *Computación Ubicua* es que, aunque con la *Computación Móvil* en algunos casos la interacción requiere interconectar dos o más dispositivos confrontados entre sí, eso no implica que se requiera ningún tipo de colaboración ni de distribución de la computación entre diversos dispositivos remotos, tal y como ocurre con la *Computación Ubicua* [SL05b].

1.1.1.3. Escenarios cotidianos

Existen numerosos ejemplos de sistemas reales que ilustran esta multiplicidad de entornos a donde poder llevar la interacción. A continuación se presentan algunos casos representativos:

- un jefe de obra provisto de una PDA, verificando sobre el terreno las medidas del plano registradas en una base de datos remota.
- arqueólogos registrando las observaciones sobre sus hallazgos directamente en el yacimiento [RN01].
- un visitante de la ciudad de Lancaster (Reino Unido) que busca en su PDA un restaurante vegetariano abierto ese día de la semana, cercano a la zona donde se encuentra [CDME01]. Ese mismo sistema es capaz de ofrecer información de interés para el usuario, conforme éste se aproxima a un punto de interés histórico.
- en la visita a museos, el kit de soporte móvil interpreta cada pieza, de acuerdo a un determinado perfil de usuario.
- en la estación, el sistema de reserva de billetes conduce eficazmente al usuario hacia el andén y zona de embarque apropiados previo al momento de partida del tren.
- un empleado de la planta de curado de jamones ibéricos supervisando el proceso de curado (Guijuelo -provincia de Salamanca-) [Gon05a].
- en el hogar, un miembro de la familia manipula una PDA que actúa como mando a distancia universal [SAW94]. El dispositivo se adapta automáticamente al aparato más cercano, bloqueándose al ser accionado por un niño.
- en la realización de diversas actividades cotidianas: transferencia de datos entre PDAs [Rek00], aumento de objetos del mundo real con facilidades computacionales [BCC00], [LSRI00], etc.

Es deseable que estas aplicaciones sean capaces de proporcionar de forma dinámica información multimedia acorde tanto al posicionamiento del usuario como a algún otro aspecto relevante del entorno. Sólo garantizando la máxima adaptación se puede ofrecer el mejor nivel de servicio al usuario para cada circunstancia, y de ese modo incrementar el *nivel de confortabilidad*[15] con el sistema.

Existen también numerosos ejemplos que ilustran cómo el *contexto de uso* (concepto definido en el *Capítulo 2*; *sección 2.1.2.*) de un mismo sistema es susceptible de cambio. Un ejemplo clásico es el sistema de control de calefacción del hogar, ideado por FEC[16] [CCT00], que permite ser controlado por usuarios situados en diversos contextos de uso. Entre ellos se encuentran: el propio hogar, a través de una Palm conectada a una red inalámbrica; en la oficina por medio de un navegador Web; o bien en cualquier lugar utilizando un teléfono móvil.

Se puede llevar más allá esta versatilidad imaginando el siguiente escenario propio de un futuro no muy lejano: cualquier usuario llegando de la calle a su casa hablando por el móvil en una red UMTS que se conmuta a la red Wi-Fi y activa la *VoIP*[17], dando paso al uso del Bluetooth al sentarse delante de un PC. Esto vislumbra la idea de lo que podría calificarse como la continuidad o ininterrupción técnica ante situaciones de migración entre dispositivos.

Si tomamos el entorno Web como referencia, es conocido por todos que la generación dinámica de documentos Web reporta grandes ventajas a la hora de mostrar al usuario datos en tiempo real, permitiendo adaptar el contenido de dicha información en función tanto de las características personales de cada usuario como de la plataforma. El acceso a Internet es, salvo excepciones, universal, a la vez que cada vez más accesible[18] para personas discapacitadas. Se puede considerar que gracias a la penetración de las aplicaciones Web se ha constatado e implantado la necesidad de considerar los perfiles y preferencias del usuario (personalización).

Típicamente, un sitio Web personalizado reconoce a sus usuarios, recoge información acerca de sus preferencias y adapta sus servicios con objeto de satisfacer sus necesidades. Se trata del campo de la *Hipermedia Adaptativa*, o también *Personalización de la Web* [Bru96]. Con la *computación móvil* se ha puesto de manifiesto que, además de la per-

[15] Se puede considerar una *IU confortable* como aquella que no sólo es fácil de usar, sino también la más apropiada para una determinada circunstancia, necesidad y condición del entorno, u otro tipo de restricción, y en cuya elección han intervenido todo o gran parte del rango de posibilidades tecnológicas que se tienen al alcance.

[16] French Electricity Company.

[17] Del inglés Voice over Internet Protocol. Voz sobre Protocolo de Internet.

[18] La accesibilidad indica la facilidad con la que algo puede ser usado, visitado o accedido en general por todas las personas, especialmente por aquellas que poseen algún tipo de discapacidad [MPI05].

sonalización, surgen otro tipo de necesidades de adaptación, como son la capacidad de acomodarse de manera autónoma y flexible al entorno físico de trabajo y demás condiciones del contexto de uso, así como también al dispositivo o plataforma de computación utilizado. En relación a este último aspecto, dada la constante proliferación de dispositivos de computación, así como la relevante heterogeneidad en sus características hardware, se precisa considerar un abanico mucho más amplio de parámetros (tamaño de pantalla, capacidades gráficas y de memoria, características de los dispositivos de entrada, posibilidades de conexión, etc.), con el fin de ofrecer el máximo rendimiento y explotación de las capacidades y funcionalidad de cada dispositivo en particular.

Esta es la idea que subyace al problema de la ***plasticidad de las IUs***, tema central de la presente tesis, definido en detalle en el siguiente capítulo.

En el *Capítulo 2* se presenta una amplia revisión de los distintos tipos de adaptación (véase *Capítulo 2*; *sección 2.1.2.*), así como también una explicación de su creciente necesidad, como antecedentes al concepto de *plasticidad* (véase *Capítulo 2*; *sección 2.1.6.*).

1.1.1.4. Precaria explotación de la interacción

Al margen de todo este avance en tecnología se evidencia, no obstante, un hecho tanto o más importante:

> *El progreso técnico no se encuentra al alcance del público, en la medida en que sería deseable.*

Ciertamente, no se ofrece toda la versatilidad de uso de la que cualquier individuo podría beneficiarse hoy en día. No se explota toda la capacidad de computación, interacción y adaptación potencial de que se dispone. Esto deriva, en ocasiones, en una escasa *confortabilidad* para el usuario, quien no ve recompensado el esfuerzo empleado en familiarizarse con un nuevo dispositivo, a cambio de una interacción pobre.

En consecuencia, existe un riesgo de descompensación entre la promesa técnica y una interacción efectiva. Suele utilizarse la expresión *"precariedad del empuje técnico"* [CCT00] para describir este hecho. Esto significa que, como sucede en muchas ocasiones, el desarrollo tecnológico está bastante más avanzado que la capacidad del sector para explotarlo y la preparación de la población para asimilarlo. La viabilidad tecnológica no es suficiente. Se requiere un esfuerzo para explotar, acercar y ofrecer al público todo el rango de posibilidades tecnológicas.

> *El gran reto no está en lo que podrá hacer la tecnología, sino en saber disponer de ella.*

Por falta de herramientas apropiadas, en general, los sistemas interactivos no funcionan más que para un solo entorno operativo prefijado de antemano, ya sea para el entorno de escritorio, el de dispositivos compactos, o el entorno Web, sin ofrecer la suficiente versatilidad como para operar libremente en cualquiera de ellos, atendiendo a las necesidades y circunstancias cambiantes, así como a las preferencias del usuario.

La multi-contextualidad se dispara bajo el efecto de la combinatoria entre la diversidad de usuarios, de plataformas y de entornos. Sería impensable que el desarrollador de una página Web tuviera que mantener manualmente una página para cada tipo de plataforma, entre ellas las de los cada vez más frecuentes dispositivos móviles con acceso a Internet. Es obvio que conviene aplicar algún tipo de mecanismo que ofrezca cierta automatización y sistematización a este proceso de generación y adaptación, no sólo de páginas Web, sino de todo tipo de IUs para sistemas interactivos.

1.1.2. Motivación

La posibilidad de llevar la interacción a contextos cada vez más dispares promete ofrecer una enorme versatilidad en la forma de acceder y consumir la información. Sin embargo, conforme aumenta esta diversidad se plantean retos cada vez más desafiantes para el desarrollador.

Las demandas de los usuarios de dispositivos móviles son cada vez más amplias, complejas y exigentes, esperando del sistema una respuesta espontánea a sus necesidades.

El público desea tener la capacidad de elegir entre un importante espectro de dispositivos para acomodar múltiples necesidades dependiendo de los espacios que se puede ir encontrando a lo largo de la interacción. Esta libertad en la elección debería existir no sólo al iniciar el uso del sistema, sino en distintos momentos de su interacción, según sean las condiciones externas, el tipo de tarea a realizar o cualquier otro parámetro o preferencia del usuario, sin que por ello se altere el contexto de la computación, a pesar de cambiar de dispositivo en el curso de la actividad. Un ejemplo sería que, ante una situación de batería baja en el ordenador portátil, el usuario continuara su tarea en una PDA, con la mínima repercusión en el contexto de su tarea. De manera similar, otro usuario puede desear mover su tarea desde una PDA a una pizarra electrónica de tipo mural, con objeto de compartir información de forma eficiente al llegar a un punto de encuentro. Estos ejemplos demuestran que la adaptación de los sistemas interactivos a los cambios contextuales se está convirtiendo en un tema de máximo interés en la comunidad de Interacción Persona Ordenador[19] (IPO).

[19]La Interacción Persona-Ordenador es una disciplina que tiene como objetivo el Diseño, Implementación y Pruebas de sistemas interactivos para el uso humano.

Para responder a esas demandas, sin duda desafiantes, hay que aceptar y afrontar que las restricciones de tiempo real relacionadas con las limitaciones en recursos hardware (ancho de banda, disponibilidad del servidor, capacidad de cómputo, restricciones de batería, etc.), las condiciones del entorno (posicionamiento, orientación, luz ambiental, día y hora, etc.), las relativas a las necesidades del usuario (nivel de evolución de sus tareas, preferencias y necesidades incrementales), y por último, las relativas al grupo de trabajo, en el caso de que se trate de una actividad colaborativa (estado y restricciones del grupo, contexto distribuido, etc., tal y como se describe en el *Capítulo 2* -véase *Capítulo 2*; *sección 2.1.3*-), son volátiles e inestables, y requieren de sofisticadas capacidades adaptativas. Denominamos a estas restricciones fluctuantes *restricciones de tiempo real*, cuya definición aparece formulada en el *Capítulo 2* (véase *sección 2.1.3*).

La caracterización de escenarios donde confluyen múltiples parámetros interrelacionados (restricciones de recursos hardware, tipo de tarea, perfil de usuario, condiciones de entorno y restricciones de grupo), se corresponde con el concepto de **contexto de uso** (concepto definido en la *sección 2.1.2* del *Capítulo 2*).

Somos testigos hoy en día de la creación de un tejido computacional extremadamente amoldable que debe afrontar nuevos requisitos en términos de procesos y de herramientas de desarrollo.

> *Es evidente reconocer que con este avance tecnológico impera una nueva constante ineludible: la* diversidad en el contexto de uso.

Esto se traduce en la necesidad de desarrollar entornos de interacción extremadamente diversificados, donde la frontera entre lo familiar y lo profesional, lo privado y lo público se hace difusa. Éstas son, precisamente, las constantes de esta tesis: afrontar y abordar parte de esta diversidad contextual, con objeto de ofrecer al público sistemas interactivos provistos de mecanismos capaces de resolver los cambios contextuales de manera automática y lo más natural posible.

Se impone flexibilizar el desarrollo y variabilidad de este tipo de sistemas, propiciando su reutilización en una amplia gama de dispositivos que utilizan técnicas de interacción distintas, consiguiendo que las IUs se acomoden a ellos. Hoy en día, a la hora de desarrollar una aplicación determinada, es conveniente plantearse también su despliegue computacional, desde dispositivos compactos a grandes ordenadores, abarcando, si procede, desde el entorno tradicional de sobremesa a los entornos ubicuos.

En esto radica el propósito de la ***plasticidad de la IU***: conseguir que el usuario se sienta lo más *confortable* posible con la IU y con el proceso de interacción con el sistema bajo cualquier circunstancia, ofreciendo solución a la necesidad de adaptación a múltiples

y constantes fuentes de variaciones, sacándole el máximo partido a las posibilidades tecnológicas que se tienen al alcance. Esto significa poder ofrecer los recursos de computación más apropiados para cada tipo de entorno y finalidad de uso.

Con objeto de proveer de *plasticidad* (concepto que se define en detalle en el Capítulo 2 (véase *Capítulo 2*; *sección 2.1*) a los sistemas interactivos, se requiere dotar a las herramientas de desarrollo de IUs de un soporte adecuado para cumplir también con este nuevo requisito. El hecho de que un sistema interactivo pueda funcionar correctamente en un amplio espectro de dispositivos y circunstancias, demostrando además la capacidad de adaptarse a los cambios contextuales sin degradar la usabilidad, proporciona sin duda un valor añadido que irá haciéndose cada vez más imprescindible.

Aunque las propiedades y técnicas desarrolladas hasta ahora en IPO [IFI97] proporcionan una base sólida, no cubren los aspectos relacionados con la variación de contextos de uso [CCT00]. Por ejemplo, no expresan la necesidad de continuidad cuando se producen transiciones entre distintos contextos de uso[CCT+02b]. En consecuencia, la información dependiente del contexto se pierde en el proceso de desarrollo debido a la falta de notaciones de diseño y de herramientas de desarrollo apropiadas. Por lo tanto,

> *"Se requieren refinar las metodologías y la forma de proceder en la disciplina de IPO para afrontar las nuevas situaciones ofrecidas por la tecnología.* ***Se necesita un modelo de proceso que integre todos estos aspectos y soporte el desarrollo de IUs plásticas****, en un intento por evitar que los principios y teorías desarrolladas hasta ahora en IPO se pierdan en la evolución de la tecnología" [CCT00].*

1.1.2.1. Retos

Podemos resumir y enunciar los retos más significativos planteados para cumplir el cometido que plantea la *plasticidad*, objeto de estudio en la presente tesis, en los dos puntos siguientes:

1. Creciente complejidad en el diseño y desarrollo de sistemas interactivos susceptibles de manifestar cambios en la IU, teniendo en cuenta multitud de factores.

Reto planteado por la existencia de una amplia gama de dispositivos en continua proliferación. Los desarrolladores de este tipo de sistemas deben afrontar la construcción de múltiples versiones para sus IUs, teniendo en cuenta multitud de factores, además de resolver la migración de un mismo sistema a distintas plataformas. Se complica, asimismo, el manejo de la configuración.

El reto radica en conseguir este objetivo sin recurrir a un desarrollo masivo. Si se optara por construir de antemano tantas versiones de IUs como contextos de uso previsibles, se

caería en un proceso extremadamente repetitivo que claramente repercutiría en detrimento de la usabilidad, así como en una explosión del coste de desarrollo y mantenimiento. Se trata de resolverlo de forma automática o semi-automática y, ante todo, sistemática, conforme se presentan nuevas situaciones.

Es por ello que debe haber un marco de desarrollo de IUs capaz de ofrecer una solución a los diversos tipos de variaciones contextuales, conforme éstas se vayan produciendo. Un aspecto de importancia capital es que se debe abordar este reto sin descuidar los aspectos de usabilidad.

Para afrontar y ofrecer esta versatilidad, es indiscutible que el sistema interactivo debe ser construido y preparado para cumplir ese requisito desde las primeras etapas de desarrollo.

2. Acentuación de la necesidad de adaptar dinámicamente la IU ante un conjunto de restricciones de tiempo real.

Si se pretende afrontar la diversidad y una máxima adaptación también durante el uso del sistema, las capacidades adaptativas deben ser incrementales, es decir, deben evolucionar en tiempo de ejecución conforme las condiciones contextuales varían, ofreciendo una respuesta dinámica a los cambios del entorno o estados posibles y previsibles del usuario. Esto pone de manifiesto una nueva capacidad: la de detectar el contexto y reaccionar de forma proactiva ante ciertos cambios contextuales establecidos de antemano.

Esta idea se corresponde con el concepto de ***Sensibilidad al Contexto***, una de las áreas de aplicación que la *Computación Ubicua* hace posible [DFAB03], definiéndola como:

Sensibilidad al contexto (Dix et al., 03) *Aquella computación capaz de personalizar las aplicaciones al contexto físico, operacional e incluso emocional del usuario, los cuales mantienen un cambio continuo.*

Paralelamente, una ***IU sensible al contexto*** es:

IU sensible al contexto (Chen et al., 00) *Aquella interfaz que dispone de cierta capacidad para ser consciente del contexto y de reaccionar ante los cambios que éste pueda experimentar.*

1.1.2.2. Consideraciones generales

Para finalizar, esta tesis defiende la idea de que *actualmente disponemos de más tecnología de la que realmente somos capaces de asimilar*, desatendiendo el hecho de que

tras esa tecnología hay una serie de personas cuyo único objetivo es beneficiarse de ella para mejorar su calidad de vida y sacar partido de la movilidad, variedad de dispositivos y posibilidades de comunicación e incluso de colaboración que hoy en día brinda la tecnología. Paradójicamente, cada día se invierte más en mejorar la tecnología existente o en inventar nuevas posibilidades para dejar obsoleta la que hoy es puntera y, sin embargo, no se estudia cómo esta tecnología debe acercarse a las personas.

Para concluir, el campo de la *plasticidad de las IUs* encuentra su justificación en el *estudio, descripción y catalogación de la problemática derivada de la diversidad de contextos de uso, en la medida que proporciona soluciones (1) económicas y (2) potencialmente ergonómicas que facilitan la adaptación de los sistemas interactivos a esa diversidad.* (1) *Soluciones económicas* porque se provee una base para la generación automática de código, lo que favorece la reutilización de modelos y diseños de IU, reduciendo considerablemente el coste de producción y mantenimiento de código para las distintas versiones. (2) *Soluciones potencialmente ergonómicas* porque las técnicas basadas en modelos posibilitan la integración de las técnicas del *diseño centrado en el usuario*. En consecuencia, es sencillo poder implicar al usuario en el desarrollo de las IUs, con objeto de velar por los parámetros establecidos de usabilidad, siguiendo una iniciativa mixta que combina operaciones manuales y automáticas.

En conclusión, este trabajo de investigación encuentra su justificación en la falta de capacidad actual para manejar adecuadamente la diversidad de condiciones de uso y circunstancias en las que hoy en día es posible interactuar, así como en la ausencia de un marco de referencia, útiles o herramientas de software apropiadas para soportar el diseño y desarrollo de *IUs plásticas*, apropiadamente adaptadas a estas nuevas necesidades.

1.2. Contexto actual del problema

En este apartado se discute brevemente la historia de las herramientas de software para el desarrollo de IUs en general, y en particular de IUs multi-contextuales, así como también de los *sistemas sensibles al contexto* existentes. Se perfilan sus éxitos y fracasos, sus problemas y limitaciones. Esta descripción se realiza teniendo en cuenta los dos retos identificados en el apartado anterior para hacer frente al cometido que plantea la *plasticidad*.

1.2.1. Creciente complejidad en el diseño de las IUs

Aunque los principios del *diseño centrado en el usuario* [IFI97] y el enfoque y técnicas de modelado proporcionan una base sólida, hasta ahora el desarrollo de IUs ha sido generalmente manual y orientado hacia un único contexto prefijado de antemano. Como consecuencia de la creciente complejidad de los sistemas software, y de manera indirecta de sus IUs, se requieren herramientas de diseño de IUs con un mayor nivel de abstracción. Actualmente, con la creciente heterogeneidad de plataformas y dispositivos de computación que aportan movilidad en el desempeño de actividades interactivas, se impone la necesidad de incrementar aún más el nivel de abstracción en el diseño y desarrollo de las IUs, en busca de mecanismos capaces de adecuar automáticamente un mismo diseño a plataformas distintas, teniendo en cuenta también los parámetros contextuales.

Descartando el enfoque tradicional, consistente en escribir versiones distintas para cada modalidad de interacción, se hace necesario migrar hacia nuevas metodologías que flexibilicen el proceso de desarrollo de IUs, a las que se les califica comúnmente como *metodologías genéricas* [SL05a]. La esencia de estos enfoques consiste en obtener especificaciones de la IU a un alto nivel de abstracción, con el propósito de evitar un desarrollo masivo de IUs. Estas consideraciones, aún antes de ser formalizadas, motivaron la aparición de nuevos conceptos como *modelo de interfaz*[20], descripción de *IU genérica*[21], etc., con la intención de ofrecer una representación conceptual de la interfaz. La idea subyacente no es otra que la de recurrir a técnicas para especificar las IUs en independencia del dispositivo, tal y como indican Myers, Hudson y Pausch en [MHP00].

Esta idea, propugnada por autores como Eisenstein, Vanderdonkt y Puerta [EVP00], ha evolucionado hacia dos aproximaciones bien definidas: (1) las *técnicas basadas en modelo*, cuyo propósito consiste en explotar toda la información modelada para producir IUs de forma sistemática, proporcionando la base para la generación de código; y (2) el uso de un *lenguaje de IU independiente de dispositivo*, normalmente basado en XML.

En particular, el *enfoque basado en modelos*, que constituye la tendencia actual más ampliamente adoptada para afrontar este problema, ha dado lugar a un extenso repertorio de técnicas y herramientas a lo largo de la década de los 90 [PP00]. Se ofrece una descripción detallada del mismo en el *Capítulo 3* (véase *Capítulo 3*; *sección 3.1.5.*). Este enfoque ofrece numerosos beneficios en el desarrollo de IUs abstractas, aunque también presenta aspectos que deben ser mejorados. Todo ello es analizado en ese mismo capítulo.

[20] Descripción formal, declarativa e independiente de dispositivo de la representación conceptual de la IU, la cual debe ser expresada a través de un lenguaje de modelado. Incluye los modelos declarativos, la IU genérica y la IU específica

[21] Expresión canónica de la renderización (representación) de las funciones y conceptos del dominio expresada de forma independiente de los objetos de interacción disponibles en cada plataforma.

Algunos de sus problemas y limitaciones identificados en 1996 por Schlungbaum en [Sch96] son: (1) la complejidad de los modelos y sus notaciones; (2) las diferencias múltiples y significativas en rango, naturaleza y notación de los modelos soportados, lo cual dificulta la comparación y reutilización de los mismos; (3) la falta de información semántica en los modelos; (4) la dificultad para modelar las relaciones entre los modelos (el conocido *problema del mapeo*); (5) el hecho de que soportan mayoritariamente la generación de IUs basadas en formularios[22] [Sch96]; (6) la dificultad de integrar las distintas IUs con la aplicación subyacente; y (7) el problema de la flexibilidad en la adaptación de contenidos y coherencia en entornos heterogéneos [LL02b]. La mayoría de estos problemas han sido centro de atención por parte de la comunidad investigadora desde el análisis realizado por Schlungbaum en [Sch96]. Fruto de ese esfuerzo se produce una mejora significativa en todos estos aspectos. En especial en los problemas identificados en (4) y (5). Así, se puede nombrar el trabajo de Montero et al. en [MLV+05], donde se proponen varias soluciones al *problema del mapeo*. La problemática del *enfoque basado en modelos* se analiza más detalladamente en el *Capítulo 3* (véase *Capítulo 3*; *sección 3.2.5*).

No obstante, a pesar del gran avance conseguido con el *enfoque basado en modelos*, no resulta suficiente para dar solución al problema de la *plasticidad*. Existen diversos aspectos que deben seguir mejorando. En la presente tesis se considera que el conjunto de modelos que se utilizan es limitado. En particular, el modelo contextual debería ser también considerado y apropiadamente relacionado con el resto de modelos, participando activamente en el proceso de construcción de *IUs plásticas*. El manejo de los aspectos del contexto también en la fase de diseño puede contribuir a la anticipación a los cambios contextuales posteriormente en la fase de ejecución, aspecto que generalmente queda sin resolver. Este es el caso, por ejemplo, de ARTStudio (Adaptation by Reification and Translation) [The01] (véase *Capítulo 3*; *sección 3.3.3*), desarrollado en el seno del grupo de investigación *IIHM*[23], (grupo que constituye un referente mundial en el campo de la *plasticidad*). Se trata de un generador semi-automático de IUs multi-contextuales. En su versión actual, ARTStudio sólo resuelve variaciones del tamaño de pantalla.

En cuanto a los *lenguajes de IU independientes de dispositivo*[24], a pesar de que existen en la actualidad diversos lenguajes de especificación de IUs con un mismo propósito de separar el diseño conceptual -el propósito o lógica de la IU- de su presentación concreta, no existe en la actualidad un lenguaje completamente independiente de dispositivo que integre todos los aspectos de la adaptabilidad de contenidos y que tenga en cuenta todo el trabajo de investigación sobre el campo, tal y como se expresa en [LL02b]. Además, lo

[22] IUs extremadamente simples, limitadas a un conjunto específico de objetos de interacción.
[23] Ingénierie de l'Interaction Homme-Machine, de la Universidad Joseph Fourier.
[24] Puede encontrarse información diversa sobre este tipo de lenguajes en [Cov01].

habitual es que cada lenguaje utilice un software específico, lo que aleja esta alternativa de una solución sistemática. La gran variedad existente, más que enriquecer reporta dificultad en la elección.

Para finalizar este análisis relativo al primero de los retos planteados, se puede decir que, en general, se evidencia una carencia de lenguajes y herramientas de soporte adecuados, quedando todavía pendientes muchos problemas sin resolver [Sch96]. Muchos de estos problemas se deben al bajo nivel de abstracción en el cual se llevan a cabo las decisiones de diseño de la IU, dando lugar a una visión de conjunto casi inexistente en esta fase. Esta es la principal limitación con la que los diseñadores y desarrolladores de IUs se encuentran. En resumen, la práctica actual en el campo de desarrollo de IUs consiste en desarrollar soluciones adhoc con pocos conceptos reutilizables en diferentes contextos. Se considera que la reutilización de modelos, métodos y herramientas que resuelvan el desarrollo de *IUs plásticas* es de vital importancia, y que sin duda debe tenerse en cuenta en el desarrollo de herramientas de soporte a la *plasticidad*.

A continuación se analiza el contexto actual relacionado con el segundo reto identificado anteriormente.

1.2.2. Acentuación de la necesidad de adaptación dinámica

Resulta obvio que para desarrollar un entorno que sea capaz de detectar el contexto, con el propósito de provocar las reacciones apropiadas en el sistema, es necesario conocer el entorno que lo rodea, a fin de caracterizarlo. Adicionalmente, se debe establecer el método para modelar e integrar dicho contexto en el funcionamiento del sistema. A pesar de que ya empieza a tomarse en consideración en el modelado de las IUs, es precisamente el aspecto de cómo utilizar efectivamente esta información lo que continúa siendo desafiante para los desarrolladores de este tipo de sistemas. Como consecuencia de ello, la anticipación a los cambios contextuales permanece aún sin resolver. De hecho, no se ha desarrollado ninguna herramienta que resuelva de forma sistemática la transición entre contextos, de tal manera que le resulte transparente al usuario [CCT01a].

El trabajo más destacado al respecto es el desarrollado en el grupo IIHM. Proponen una infraestructura para el tiempo de ejecución consistente en un mecanismo software para la detección automática de cambios contextuales basados en desviaciones en las propiedades del sistema y patrón de actuación del usuario. Se trata de Probe [CCT01a], una componente de CatchIt (Critic-based Automatic and Transparent tool for Computer-Human Interaction Testing) [Cal98].

Asimismo, estructuran el espacio de solución para el problema de adaptar la IU a los cambios contextuales mediante un proceso en tres pasos: (1) reconocimiento de la situación; (2) computación de una reacción adecuada a la nueva situación; y (3) ejecución de la correspondiente reacción. Las reacciones se especifican en un *modelo de evolución*[25], el cual es ejecutado por un *supervisor del contexto*, adaptando cuando es posible la IU al nuevo contexto de uso. A pesar de la relevancia de este trabajo, quedan abiertos gran cantidad de aspectos, entre ellos el de incluir recomendaciones o heurísticas para clasificar, ordenar y jerarquizar los entornos y plataformas. Adicionalmente, se requiere su despliegue de manera sistemática para abordar la transición entre contextos de manera transparente. Tal y como sostienen sus propios autores, esta herramienta no se encuentra todavía completamente formalizada [CCT01a].

Sin duda, la combinación de cuantas propiedades contextuales nos sea posible favorece un entendimiento más preciso de la situación en curso. Sin embargo, a parte del tratamiento de la localización y del modelado del usuario, ningún otro parámetro del contexto ha sido suficientemente estudiado [CK00]. Por otro lado, a pesar de la aparente correlación y proximidad entre el modelado de usuario y el modelado del contexto, tampoco se ha empleado demasiado esfuerzo en explorar técnicas que los combinen explícitamente, constituyendo un área de investigación relativamente inexplorada, tal y como se analiza en [BC01]. Prueba de ello es que hasta la fecha existen pocos ejemplos de aplicaciones que reúnan y conjuguen ambas técnicas [BC01]. La mayoría de las aplicaciones existentes utilizan sólo unos pocos valores del contexto (los más frecuentes son la localización, la identidad y el tiempo). Además, la ausencia de un adecuado mecanismo de propagación de cambios contextuales en el caso de que se trate de sistemas compuestos por varias componentes, constituye hoy en día una de las carencias más importantes detectadas en la literatura en este campo.

En resumen, la mayoría de las *aplicaciones sensibles al contexto* existentes son prototipos desarrollados en laboratorios de investigación en el mundo académico. Sí existen, sin embargo, diferentes tipos de servicios basados en localización [Bar03] y guías de visita de ciudades y museos, tal y como se mencionan en [CCT01a] y [Gon05a]. La mayoría de aplicaciones de este tipo han sido construidas de manera adhoc, resultando difícil la construcción de nuevas aplicaciones, o evolucionar las ya existentes. En efecto, en los enfoques basados en arquitectura, los mecanismos internos que soportan auto-adaptación son, con frecuencia, altamente específicos a la aplicación y están estrechamente ligados al código. Existe la necesidad de una infraestructura estandarizada para el intercambio de información contextual. Esta problemática se analiza con más detalle en el *Capítulo 5*.

[25]Modelo que especifica la reacción a aplicar cuando se producen cambios en el contexto de uso.

Una vez analizada la situación en lo que respecta a los dos retos identificados, se hace evidente que existen necesidades de investigación en el campo de las *IUs plásticas*. A pesar de que el trabajo del grupo IIHM de Grenoble es inestimable, todavía quedan aspectos de máximo interés por resolver. Entre los más notables está la integración de mecanismos adecuados para controlar, medir, ajustar y tener presente en todo momento los objetivos de usabilidad. El trabajo más representativo en este sentido es CatchIt [Cal98], un entorno basado en computador que combina tanto una evaluación predictiva[26] como experimental[27] de la usabilidad.

Puesto que en la presente tesis también se analiza la aplicabilidad del modelo propuesto al diseño de entornos colaborativos flexibles y heterogéneos, revisando su adecuación y tratando de especializarlo para el tratamiento de los aspectos intrínsecos al trabajo en grupo, se hace indispensable presentar también una breve descripción del contexto actual en el diseño de entornos colaborativos. La siguiente sección está destinada a ello.

1.2.3. Contexto actual en el diseño de entornos colaborativos

En el campo del CSCW[28], es ampliamente reconocida la importancia de tener en cuenta la compleja dinámica social donde la actividad grupal se desarrolla. De hecho, esta premisa es anunciada como uno de los ocho retos que debe afrontar un desarrollador de groupware, identificados por Grudin en [Gru94]. Lamentablemente, esta componente no está presente de manera sistemática en los enfoques encontrados en la literatura.

Un trabajo de grupo exitoso no surge de la simple unión de tareas individuales, sino de un conjunto organizado de actividades coherentes con buenas estrategias de comunicación, cooperación y coordinación entre los miembros del grupo, que den lugar a una interacción fluida entre todos los miembros del grupo [AGOP05]. Además, es necesario considerar el carácter de las tareas y las responsabilidades, así como de qué manera el acceso a la información requiere y suscita colaboración. Está comprobado que parte de los problemas que se detectan en ciertas aplicaciones de trabajo en grupo es debido a no haber tenido en consideración el contexto social donde se lleva a cabo la interacción, incluyendo la identificación de tendencias favorables y relaciones de coherencia entre los elementos que caracterizan la actividad colaborativa [Bar97], [Gru94].

[26] Enfoque que verifica si las propiedades generales del sistema previamente establecidas y expresadas en términos de métricas, tales como la observabilidad, son satisfechas por parte del sistema.

[27] Enfoque que proporciona realimentación acerca de la usabilidad de un sistema, detectando automáticamente posibles desviaciones. Por lo tanto, es capaz de detectar si el comportamiento del usuario actual se corresponde con las expectativas de los diseñadores, ejecutando las acciones esperadas.

[28] De inglés Computer Supported Cooperative Work.

El proceso de diseño de una aplicación colaborativa debería estar centrada en el contexto con objeto de asegurar que la herramienta en cuestión será útil para soportar la actividad colaborativa para la que se está construyendo. La falta de consideración de estos elementos contextuales difícilmente permite desarrollar una herramienta colaborativa adecuada a la actividad grupal en cuestión. Puede encontrarse una descripción más detallada de toda esta panorámica en [Sen07].

En este sentido, los diseñadores de *groupware* han incluido este tipo de aspectos, conocidos como aspectos de *awareness*[29] [DB92] –entendimiento colaborativo, que suele traducirse como *consciencia de grupo*-, destacando la importancia de soportarlo, aspecto que constituye un pilar en el diseño de sistemas *groupware*. Se trata de un concepto de diseño que contribuye a reducir el esfuerzo meta-comunicativo necesario para desarrollar actividades colaborativas o *sistemas distribuidos en el espacio*[30], promoviendo una colaboración real entre los miembros del grupo [PR96]. Muchos enfoques de *groupware* consideran estos aspectos proporcionando distintos *mecanismos de consciencia de grupo*[31]. Sin embargo, el contexto compartido es poco estructurado, implícito y restringido a cada aplicación. A pesar de su relevancia, no se ofrece un soporte sistemático y los desarrolladores deben comenzar siempre desde cero en cada nuevo sistema. Debe realizarse un gran esfuerzo para mejorar y sistematizar el desarrollo de soportes para esta clase de información.

Por otro lado, la forma en la que los mecanismos de *awareness* son soportados constituye también un factor relevante a ser considerado. En la mayoría de las arquitecturas, como por ejemplo en la propuesta por el grupo de Favela del centro de investigación CICESE[32] [FRPG04], los aspectos de grupo son manejados mediante la comunicación entre un conjunto de agentes en el lado del servidor y un conjunto de monitores en el lado del cliente. Como consecuencia, la información está centralizada, lo que conlleva a un enfoque excesivamente centrado en el servidor. Los mecanismos de *awareness* están demasiado condicionados al estado de la red, elevando considerablemente el coste de proporcionar *awareness* e impidiendo la autonomía en los dispositivos locales.

La alternativa propuesta en la literatura consiste en utilizar un enfoque totalmente distribuido, persiguiendo autonomía y flexibilidad. Un ejemplo representativo es el de los módulos de software SOMU, donde cada dispositivo alberga un complejo middleware capaz de proporcionar y consumir servicios de otras unidades o dispositivos, utilizando

[29] Entendimiento de las actividades de los otros integrantes de un grupo de trabajo, que proporciona un contexto para la propia actividad [DB92].

[30] Actividades colaborativas donde los integrantes del grupo de trabajo ocupan distintas localizaciones (posiciones geográficamente distribuidas) [EGR91].

[31] Del inglés *awareness mechanisms*. Cualquier mecanismo de soporte en la construcción y mantenimiento del conocimiento relativo a la actividad grupal y que contribuya a su entendimiento y por tanto a una mayor efectividad.

[32] http://www.cicese.mx/

Programación Orientada a Servicios[33] [NOP06]. Este enfoque tiene ciertas limitaciones, como son el manejo de la información compartida y la coherencia, las cuales constituyen un problema no trivial, al implicar distintas problemáticas asociadas a aspectos como la replicación de la información.

En conclusión, la mayoría de los enfoques revisados son o bien centralizados o bien completamente distribuidos, no existiendo una alternativa intermedia más o menos equilibrada [NOP06].

Una vez realizado este análisis sobre el contexto actual del problema la sección siguiente está destinada a la formulación de las hipótesis de partida, teniendo en cuenta parte de las limitaciones detectadas.

1.3. Hipótesis de investigación

La formulación de las hipótesis de investigación planteadas en la presente tesis conlleva varias consideraciones. Por este motivo, y con el fin de facilitar su comprensión, se presentan agrupadas en diversos apartados temáticos.

Enfoque y distribución de responsabilidades.

Con el fin de poder afrontar la doble problemática identificada en torno a la *plasticidad* de manera adecuada y en su totalidad, se cree necesario proporcionar un soporte apropiado y específico para cada reto formulado, los cuales incluyen unas estrategias y técnicas de modelado e implementación distintas. Por lo tanto, esta primera hipótesis se enuncia de este modo:

Hipótesis 1 (Enfoque de plasticidad) *La realización de un tratamiento por separado de la doble problemática identificada, enmarcándola bajo un modelo de arquitectura cliente-servidor, permite alcanzar un equilibrio operacional entre ambas componentes (cliente y servidor) que optimiza el proceso de adaptación.*

Es ampliamente reconocido que para automatizar la reconfiguración y generación automática de IUs (propósito asociado al primer reto) se requiere un soporte capaz de estructurar y automatizar la producción de *IUs plásticas* basado en la especificación a un alto nivel de abstracción de la IU, inspirado en las *técnicas basadas en modelos*[Lóp05]. Este tipo de herramientas alivian el coste de desarrollo y mantenimiento.

[33] Un nuevo paradigma de programación que gana popularidad en entornos de computación distribuidos debido a su enfoque hacia código altamente especializado, modular e independiente de plataforma, que facilita la interoperabilidad de los sistemas.

Sin embargo, para automatizar una respuesta proactiva (anticipativa) a los cambios contextuales no existe ningún enfoque estándar, ni tampoco se ha consensuado ningún tipo de solución universal, a pesar de encontrar diversas aproximaciones a lo largo de la literatura sobre el tema.

En esta tesis se plantean las dos siguientes hipótesis al respecto:

Enfoque tecnológico para soportar adaptaciones proactivas.

Hipótesis 2 (Ejecución de adaptaciones proactivas) *La ejecución de las adaptaciones proactivas en el propio equipo donde se lleve a cabo la interacción (incluso tratándose de dispositivos limitados tipo teléfono móvil), sin interceptar en la operativa del sistema ni en la actividad del usuario, reporta numerosos beneficios, entre ellos reducir el nivel de dependencia de los recursos de red, limitando la comunicación a situaciones contextuales que no puedan ser asumidas localmente.*

Se estima que la integración de los mecanismos necesarios para ofrecer una respuesta proactiva a ciertos cambios contextuales de naturaleza dinámica en el propio sistema software genera código *disperso*[34] y *enmarañado*[35]. Los *requisitos software auxiliares*[36] que presentan esta problemática, y cuya separación en componentes independientes resulta muy difícil, se denominan ***conceptos transversales***[37]. Por lo tanto, esta tercera hipótesis se enuncia de este modo:

Hipótesis 3 (Implementación de las adaptaciones proactivas) *El manejo de los diversos factores contextuales y sus respectivos mecanismos de adaptación proactiva generan conceptos transversales inherentemente relacionados con los requisitos funcionales en su integración con el sistema software.*

Debe aplicarse alguna técnica de *separación de conceptos*[38] para embeber en el sistema los mecanismos de adaptación proactiva. Se asume que la técnica de separación de conceptos más adecuada, con el fin de obtener unos cánones de calidad basados en la independencia entre los distintos requisitos, la transparencia para el sistema en la manera

[34]Código redundante y esparcido a lo largo de diversos módulos del sistema; a veces denominado código entrecruzado.

[35]Código encargado de afrontar diversos requisitos, generalmente sin relación aparente entre ellos, entremezclándose el tratamiento de los mismos.

[36]requisitos extra-funcionales, en ocasiones referidos también como adicionales.

[37]Del término inglés "crosscutting concerns".

[38]Del inglés *separation of concerns,* cuyo centro de atención es la modularización de los requisitos auxiliares, fomentada por la Ingeniería del Software.

como son integrados y su reutilización en diferentes sistemas es la técnica de Programación Orientada a Aspectos, por su facilidad para modularizar *conceptos transversales*. Por lo tanto, esta cuarta hipótesis se enuncia de este modo:

Hipótesis 4 (Requisitos de diseño y estructura software) *La técnica de Programación Orientada a Aspectos favorece la integración de los mecanismos de adaptación proactiva en la operativa del sistema bajo unos cánones de calidad relacionados con las propiedades de ortogonalidad, transparencia y reutilización, los cuales reportan numerosos beneficios, entre los que se presupone la posibilidad de ofrecer un* ***framework genérico***[39]*.*

Hipótesis 5 (Contribución desde la fase de diseño) *El manejo de los aspectos del contexto también en la fase de diseño contribuye a una anticipación a los cambios contextuales durante la fase de ejecución, no sólo ofreciendo una IU adecuada a las nuevas condiciones, sino también facilitando los mecanismos de adaptación proactiva más apropiados para afrontar la situación en curso.*

Realimentación en el proceso de plasticidad.

Hipótesis 6 (Mecanismo de propagación de cambios contextuales)
El mecanismo de comunicación subyacente a cualquier arquitectura cliente-servidor ofrece el soporte necesario para la propagación de cambios contextuales que se producen en la plataforma cliente hacia el servidor, facilitando un proceso iterativo y alternativo supeditado a las necesidades de adaptación que puedan ir surgiendo a lo largo del proceso de interacción, el cual permite complementar ambas componentes manteniéndolas en continua actualización.

Hipótesis 7 (Integración y explotación de la información de awareness)
La infraestructura y el modelo de plasticidad propuestos permiten incorporar y explotar aspectos intrínsecos al trabajo en grupo en el propio proceso de plasticidad, proporcionando y combinando convenientemente dos niveles de consciencia de grupo, que corresponden a los dos niveles de plasticidad identificados, basados en la doble problemática y la arquitectura asumida en la Hipótesis 1.

[39] Conjunto cohesivo y extensible de clases que colaboran para proporcionar la parte central e invariable de un subsistema lógico, destinado a una determinada funcionalidad o servicio.

1.4. Objetivos generales

Siguiendo con el enfoque de estudiar por separado las herramientas específicas para abordar las dos metas identificadas previamente (véase con más detalle en *Capítulo 2;* sección 2.2.1), los objetivos generales de la investigación llevada a cabo en esta tesis se pueden sintetizar en los dos siguientes:

- Proponer un medio para expresar información dependiente del contexto tanto en tiempo de ejecución como en tiempo de diseño.
- Proponer una infraestructura software basada en el modelo cliente-servidor para dar soporte a la *plasticidad* tanto en la fase de diseño de la IU como en la fase de ejecución, centrando todo el estudio y desarrollo en algunas limitaciones y problemas encontrados en la literatura, los cuales ya han sido tratados superficialmente en la *sección 1.2* del presente capítulo. El *leitmotiv*[40] es el de conseguir distribuir la responsabilidad de adaptación a ambos lados de una arquitectura cliente-servidor, de manera flexible.

En particular, en relación a este último punto, y tal y como se desglosa en la sección siguiente, en el lado del servidor de la arquitectura propuesta se propone abordar la parte relativa a la fase de diseño, es decir, la asociada al primer reto identificado (véase *Capítulo 2; sección 2.2.4.2*). Para ello se aporta un estudio de los marcos conceptuales de referencia existentes en la literatura para la construcción de soportes de desarrollo de IUs de estas características, capaces de estructurar y automatizar su producción. Con su estudio y caracterización se pretende proponer un nuevo modelo conceptual teniendo en cuenta los resultados del análisis realizado. Como características más destacadas se propone abordar estos dos requisitos: (1) preservación en todo momento de la usabilidad; y como aspecto novedoso (2) la integración en el marco conceptual de componentes propios de entornos colaborativos, es decir, incorporando aspectos intrínsecos del trabajo en grupo.

Complementariamente, en el lado del cliente se propone abordar la parte relativa a la fase de ejecución, es decir, el desarrollo de unos mecanismos para detectar el entorno y desencadenar la reacción oportuna para producir los correspondientes reajustes en la IU. De este modo se aborda el segundo reto identificado, relativo a una adaptación proactiva. Esta meta implica ofrecer una solución efectiva a todos estos aspectos: (1) caracterizar el contexto; (2) modelarlo; (3) integrar dicho modelo en la operativa del sistema; y finalmente (4) utilizar efectivamente la información contextual para determinar qué cambios deben ser practicados sobre la IU.

[40] De *Leiten*, guiar, dirigir, y *Motiv*, motivo: motivo conductor

Adicionalmente, una vez formulada una propuesta de solución para la adaptación a abordar en el lado del cliente, se plantea el desarrollo de un *framework genérico* que, siguiendo unas directrices de abstracción, permita reutilizar código en la construcción de *aplicaciones sensibles al contexto*. La meta perseguida es la de facilitar el diseño e implementación de soluciones *sensibles al contexto* válidas para diferentes dominios de aplicación.

1.5. Objetivos específicos

Los objetivos específicos planteados en la presente tesis son los siguientes:

- Identificar, delimitar y acotar los dos niveles de operación detectados en el proceso de *plasticidad*, asociados a los dos retos presentados en la sección introductoria de esta tesis, como visión particular de la problemática. Asignar respectivamente dos sub-conceptos y dos metas claramente diferenciadas.
- Proporcionar un mecanismo de propagación de cambios contextuales y realimentación consecuente entre todos los componentes que intervienen en el marco arquitectural propuesto; para ser más precisos, entre ambos lados de la arquitectura cliente-servidor. Esto permitirá que el servidor produzca IUs lo más ajustadas posible a la situación del momento.
- En relación con el aspecto colaborativo, se pretende especializar los motores de *plasticidad* propuestos en esta tesis con el fin de integrar la información de grupo y explotarla como parte integral del proceso de *plasticidad*. La idea es incluir todos los aspectos relativos a *plasticidad* y *awareness* en un framework integrador, aplicando un enfoque centrado en el contexto y proporcionando *mecanismos de consciencia de grupo*.

En consecuencia y en sintonía con la doble problemática identificada, se exponen a continuación los objetivos planteados desde cada una de las perspectivas.

Problemática asociada al primer reto: "Creciente complejidad de diseño de las IUs".

- Trazar un marco conceptual que sirva de referencia con el fin de proporcionar una orientación para la construcción de herramientas de diseño y desarrollo (semi)-automático de *IUs plásticas*, que además soporte el trabajo colaborativo. Al igual

que otros marcos conceptuales o modelos de referencia, también se propone como instrumento de referencia para el estudio en profundidad, comparación y razonamiento acerca de herramientas ya existentes.

- Determinar el conjunto de modelos más adecuado para soportar la variabilidad del contexto, incluyendo el caso de entornos colaborativos. En definitiva, proporcionar las directrices para definir un modelado conceptual de la interfaz suficientemente completo, así como una utilización efectiva del mismo.

Problemática asociada al segundo reto: "Acentuación de la necesidad de adaptación dinámica".

- Proponer una arquitectura software, juntamente con los mecanismos asociados para resolver la anticipación a los cambios en aquellos atributos del contexto de naturaleza dinámica previamente establecidos. Este objetivo comprende la capacidad de detectar el contexto, deducción de la reacción a tomar por el sistema, y finalmente ejecutarla produciendo una adaptación de la IU. Para ser más precisos, se trata de dotar a las IUs de la capacidad de soportar variaciones en el contexto de uso, embebiendo esos mecanismos en el propio sistema interactivo.

- Ofrecer sistematización, reutilización y flexibilidad en la manera en que se programa la anticipación a los cambios contextuales. Para ello se pretende ofrecer unas directrices de abstracción, reutilización y extensibilidad para la construcción y futura amplicación de un *framework genérico* fácilmente *personalizable*[41] a sistemas concretos con necesidades particulares, así como un método para su manejo y explotación ya desde la fase de diseño.
  Para facilitar su uso y despliegue lo ideal sería organizar y materializar dicho framework a través de una librería jerárquica de elementos de programa o módulos genéricos, de fácil integración en los códigos núcleo de las aplicaciones, y convenientemente clasificada. En concreto, se pretende conseguir su reutilización a diferentes familias de sistemas, diferentes necesidades contextuales y diferentes mecanismos de adaptación.

1.6. Clasificación de la tesis

La presente tesis se puede considerar enmarcada en la intersección entre la Ingeniería del Software y la Interacción Persona Ordenador. Se trata del campo de dominio que en

[41] El sello de calidad de un *framework* es precisamente que proporcione una implementación para las funciones básicas e invariables, e incluya un mecanismo que permitan al desarrollador conectar las funciones que varían, o extenderlas.

ocasiones se ha denominado *"Software Engineering domain of Human Computer Interaction"*, o simplemente *"Engineering for Human-Computer Interaction"*; un campo que puede considerarse como la *Ingeniería del Software al servicio de la IPO*, ofreciendo el soporte necesario para la construcción de sistemas interactivos e IUs con requisitos adicionales que van más allá del diálogo entre el hombre y la máquina. Este campo abarca el estudio y desarrollo de conceptos, modelos, estilos arquitectónicos y herramientas software para el diseño, implementación y evaluación de sistemas interactivos, tal y como aparece en la Web oficial del grupo *IIHM*. En síntesis, se podría caracterizar este campo de dominio como aquel cuyo propósito es el de aplicar ciertas especialidades de la Ingeniería del Software en beneficio de la Interacción Persona-Ordenador (en adelante IPO) para idear, inventar y construir *artefactos software*[42] o herramientas útiles para el campo de la IPO. Así es como también se enmarcó el proyecto Europeo CAMELEON (Context Aware Modelling for Enabling and Leveraging Effective interactiON)[43], uno de los proyectos más importantes sobre *plasticidad*.

En efecto, tal y como se ha planteado en la sección de objetivos concretos asociados al segundo reto planteado en la presente tesis, se pretenden idear mecanismos basados en algún tipo de soporte o arquitectura software para su integración en sistemas interactivos siguiendo los cánones de calidad del software. Para llevar a cabo este propósito es ineludible considerar y hacer prevalecer en todo momento los principios fundamentales de diseño del software. La intención es la de proveer a los sistemas interactivos con capacidades de adaptación. De hecho, se persigue armonizar la intervención de tres componentes: el humano, el equipo de computación y el entorno que rodea la actividad, con el fin de acomodar una interacción conjunta.

Según Sommerville, la *Ingeniería del Software* es la disciplina de ingeniería que comprende todos los aspectos de producción de software [Som05]. Entre los numerosos retos que esta disciplina afronta, aquellos con los que se identifican los objetivos de la presente tesis son los de siguientes: (1) el desarrollo de herramientas, métodos, teorías, *frameworks* y arquitecturas software de apoyo a la producción de sistemas software en general, y sistemas interactivos en particular; y (2) conseguir reutilización en los productos software obtenidos.

En particular, la disciplina de la IPO tiene como meta la creación de nuevos métodos, técnicas y marcos de desarrollo para conseguir acomodar en el mayor grado posible la interacción entre la máquina y el ser humano que la utiliza, procurando que aquélla sea más

[42] Uno de muchos tipos de materiales o documentos tangibles o piezas de información utilizados o producidos durante el desarrollo de software.

[43] IST-R&D : Plasticity of User Interfaces; 2001-2004. http://giove.cnuce.cnr.it/cameleon.html

receptiva a las necesidades de los usuarios [Nie93], meta con la que se identifica la presente tesis. En otras palabras, la IPO se propone minimizar la barrera entre el modelo cognitivo del humano acerca de aquello que se propone conseguir y la consecuente mecanización o computerización en una tarea concreta soportada por ordenador. Se trata de dar soporte, en definitiva, al diseño de sistemas interactivos usables.

Esta disciplina está despertando un creciente interés en los últimos años. En nuestro país, la Asociación Española de Interacción Persona-Ordenador[44] (AIPO), con ocho años de antigüedad, está desarrollando un papel crucial, promoviendo la investigación científica de la disciplina y facilitando que los profesionales de nuestro país aporten su experiencia al campo académico.

Puesto que la finalidad de la presente tesis es la de conseguir que la IU sea capaz de amoldarse a distintas condiciones y situaciones contextuales, con el propósito de proporcionar en todo momento la IU más apropiada para cada situación, es obvio que se está contribuyendo al campo de la IPO. Otro aspecto esencial que no debemos descuidar es que el concepto de la *plasticidad* está estrechamente ligado con el concepto de *usabilidad*, tal y como se detalla en el Capítulo siguiente (véase *sección 2.1.1*). Cabe decir también que desde el acuñamiento del término de *plasticidad* en 1999, éste siempre se ha vinculado al campo de la IPO, tal y como se explica en el próximo capítulo (véase *sección 2.1*).

Todas estas reflexiones refuerzan la visión de que la presente tesis se sitúa en la intersección de la IPO con la Ingeniería de Software. De hecho, esto no resulta sorprendente tratándose de una disciplina como la IPO, eminentemente interdisciplinar, tal y como es ampliamente reconocido [AAC+02]. Nos atreveríamos a llamar a esta área *"la vertiente ingenieril que dirige la construcción y gestión de IUs desde el punto de vista de la Ingeniería de Software"*.

Por último, tal y como se ha comentado anteriormente, en la presente tesis se analiza la aplicabilidad del modelo propuesto al diseño de entornos colaborativos, procurando incorporar también los aspectos relativos al grupo de trabajo en el proceso de *plasticidad*. Para ser más precisos, se manifiesta una cierta inquietud por fomentar la interacción y un entendimiento compartido entre los miembros del grupo. Esta es una de las metas que plantea todo sistema de *groupware*. Según Ellis, el campo de *groupware* se encarga de ofrecer un soporte para la interacción usuario a usuario, situando como objetivo principal de este campo el de asistir a los grupos de trabajo en la comunicación, colaboración y coordinación de sus actividades comunes [EGR91].

[44] AIPO es una asociación profesional en la disciplina de la Interacción Persona-Ordenador. http://griho.udl.es:8080/aipo/

Tanto si los miembros del grupo comparten espacio como si esa actividad conjunta se realiza a través de distancias físicas, con o sin la posibilidad de compartir tiempo (entornos colaborativos síncronos o asíncronos respectivamente), el ordenador facilitará una interfaz común para un entorno compartido con el fin de plasmar y dar a conocer a todos los miembros cualquier conocimiento que deba ser compartido por el grupo, ofreciendo un soporte para la consecución de una meta común. De cómo se plasme ese conocimiento en esa IU, así como de lo funcional y usable que resulte depende, en parte, el éxito del entorno colaborativo y, en definitiva, de la actividad grupal. No es por tanto sorprendente la contribución que puede ofrecer el campo de la IPO en el de *groupware*. Por ello, y de manera tangencial, esta tesis se puede enmarcar también en el campo de *groupware*, en particular en el diseño de entornos colaborativos, ocupando, en definitiva, un lugar en la intersección entre la Ingeniería del Software, la Interacción persona-Ordenador y el *groupware* (específicamente en el diseño de entornos colaborativos).

1.7. Estructura del documento

El presente documento está dividido en un total de 9 capítulos y 3 apéndices complementarios que se estructuran a lo largo de tres bloques temáticos. El primero de ellos consiste en una contextualización del problema objeto de estudio: la *plasticidad de las IUs*, en el que también se presenta el modelo de plasticidad propuesto en esta tesis, denominado *Visión Dicotómica de Plasticidad*. Los otros dos bloques están destinados al estudio de cada una de las sub-áreas relacionadas con cada uno de los retos identificados (denominadas en esta tesis *plasticidad explícita* y *plasticidad implícita*), y en los que también se presentan los correspondientes artefactos software desarrollados para cada ámbito.

Así, la parte de *contextualización* consta de los siguientes capítulos:

El **Capítulo 1** está dedicado a la introducción de la tesis mostrando la motivación de la misma, una descripción superficial del contexto actual del problema, las hipótesis de partida, la relación de los objetivos planteados y finalmente la clasificación de la misma.

El **Capítulo 2** explora su tema central: *la plasticidad de la IU*. En una primera parte se presenta el origen y evolución del término, así como los conceptos relacionados y sus diferentes variantes. En una segunda parte se justifica la necesidad de extender el concepto de *plasticidad* para introducir el enfoque que sustenta la presente tesis: la *Visión Dicotómica de plasticidad*. Se presentan los antecedentes de esta perspectiva, algunos aspectos tecnológicos y las ventajas asociadas.

La parte de *plasticidad explícita* se compone de los siguientes capítulos:

El **Capítulo 3** está enteramente dedicado a la revisión del estado del arte en enfoques de especificación de IUs en general, y de *técnicas basadas en modelos* en particular. Se presenta el marco de referencia unificado CAMELEON Reference Framework, desde su primer esquema de concepción hasta su estado actual. Sin duda, constituye la referencia para la definición del marco conceptual para el desarrollo de *IUs plásticas* presentado en la presente tesis.

El **Capítulo 4** describe el marco conceptual propuesto como instrumento de referencia, al que se le denomina *Framework de Plasticidad Explícita Colaborativo*. Se analizan las aportaciones del mismo con respecto al CAMELEON Reference Framework.

La parte de *plasticidad implícita* consta de los siguientes capítulos:

El **Capítulo 5** presenta una revisión del estado del arte en herramientas *sensibles al contexto*, analizando las limitaciones en el área. También se muestra una comparativa entre distintas aplicaciones y herramientas significativas en el campo.

El **Capítulo 6** describe la arquitectura software propuesta para dar soporte a la *plasticidad* en la plataforma cliente, justificando cada decisión de diseño e implementación tomada. Como la propuesta se basa en un enfoque *orientado a aspectos*, se analiza el estado del arte en herramientas de adaptación dinámica a distintos factores contextuales basadas también en este enfoque.

El **Capítulo 7** presenta en detalle los casos de estudio que han constituido la base para el diseño y desarrollo de sistemas *sensibles al contexto* bajo las directrices software y enfoques técnicos expuestos en el *Capítulo 6*, así como el escenario de experimentación acerca de la idoneidad de la técnica aplicada en la plataforma destino.

El **Capítulo 8** presenta el framework genérico para el desarrollo sistemático de componentes de *sensibilidad al contexto*, a ser embebidas en aplicaciones ya existentes, denominado *Framework de Plasticidad Implícita*. Se presenta también la integración de todos los componentes propuestos en la presente tesis, a fin de proporcionar una visión integradora de la operativa conjunta. Esta perspectiva se presenta bajo el enfoque de la *Visión Dicotómica de plasticidad* presentada en el *Capítulo 2*, la cual, como ya se ha comentado anteriormente, fundamenta toda la tesis.

Y, por último, el capítulo conclusivo:

El **Capítulo 9** sintetiza las principales conclusiones extraídas de la tesis y las aportaciones del trabajo realizado. Se presenta el trabajo futuro, identificándose nuevas líneas que complementan el desarrollo presentado. Se incluyen también las principales publicaciones derivadas de este trabajo.

Los apéndices complementan los contenidos de los capítulos de la forma siguiente:

En el **apéndice A** se presentan detalles acerca de las construcciones propias del *lenguaje orientado a aspectos* utilizado, el lenguaje *AspectJ*, así como otros aspectos generales de la *Programación Orientada a Aspectos* en términos de *AspectJ*.

El **apéndice B** proporciona los detalles de diseño e implementación para los esquemas de solución correspondientes a las cinco componentes desarrolladas en el framework, expresados de manera genérica.

El **apéndice C** recoge los diagramas de clases que representan cada uno de los componentes desarrollados del framework, complementando los que se presentan en el *Capítulo 8*.

Capítulo 2

Plasticidad de la Interfaz de Usuario

"Enjoying success requires the ability to adapt. Only by being open to change will you have a true opportunity to get the most from your talent."

Nolan Ryan

Este capítulo describe en detalle el área de estudio conocido como ***plasticidad de la IU***, como parte principal del marco teórico del trabajo presentado en esta tesis. Se trata de un área de reciente aparición perteneciente a la disciplina de la Interacción Persona Ordenador.

Primeramente se presenta el origen y evolución del término, los conceptos relacionados y sus antecedentes. A continuación, se enfatiza la descripción de la problemática que se aborda en esta tesis, justificando la necesidad de extender su concepción inicial con dos nuevos sub-conceptos, los cuales caracterizan el modelo de plasticidad propuesto en esta tesis, denominado *Visión Dicotómica de Plasticidad*. El enfoque basado en este modelo es el que fundamenta tanto el estudio llevado a cabo como la solución propuesta en la presente tesis.

2.1. Definición y Caracterización del término de *Plasticidad*

Para explicar en profundidad el problema de la ***plasticidad de las IUs***, se presenta en primer lugar la descripción utilizada para acuñar el término y la evolución que ha tomado esa primera noción como consecuencia de la trayectoria que ha seguido el trabajo relacionado por parte del grupo que le dio origen. Se trata del grupo *IIHM*[1] de la Universidad Joseph Fourier en Grenoble, el cual puede considerarse como la "cuna" de la *plasticidad*. Conforme se introducen las diversas variaciones en la definición se justifican los motivos que han dado lugar a esos cambios. Asimismo, para una completa comprensión es imprescindible describir también los conceptos en los que se basa la definición tomada como referencia.

Una vez contextualizado el término, se presentan los aspectos inherentemente relacionados, como por ejemplo el de *contexto de uso* y la caracterización que se hace del mismo en esta tesis. A continuación se presentan distintas variantes en torno a la noción objeto de estudio, cuya distinción aportará una mejor comprensión, así como otros conceptos relacionados. Para finalizar esta sección, se presentan las distintas manifestaciones de adaptación que han ido surgiendo, como antecedentes a la *plasticidad*.

2.1.1. Acuñamiento y evolución cronológica del término

Aunque el término de ***plasticidad de la IU*** fue introducido por primera vez por Coutaz en 1998 en [Cou98], no se le dio aún la dimensión apropiada, puesto que se obvió la propiedad de usabilidad. En aquel momento se presentó como

> *"la capacidad de adaptación de un sistema informático determinado, a distintas plataformas software y parámetros hardware".*

No obstante, con esta primera definición se sientan ya las bases del lema que fundamenta el concepto:"especifica una IU, genera múltiples" [Cou98], en previsión de todos los cambios tecnológicos que por entonces ya se auguraban. En este primer eslogan se pone ya de manifiesto que la *plasticidad* trata de evitar tener que construir una a una diversas IUs para distintos estilos de interacción, vislumbrando la necesidad de 'especificar' las IUs, más que la de 'escribir código ejecutable', como mecanismo de abstracción de las posibles variaciones que pueda sufrir la interfaz. Es evidente que la meta a alcanzar va más allá de un mero objetivo de portabilidad[2].

[1] Ingénierie de l'Interaction Homme-Machine, de la Universidad Joseph Fourier.

[2] cuyo eslogan es: "codifica una vez, ejecuta múltiples". En la *sección 2.1.4* se exponen con más detalle las diferencias.

No es hasta 1999 que Thevenin y Coutaz ofrecen una nueva descripción de *plasticidad de la IU* en la que se imprime ya el sello de la usabilidad [TC99], dándole la relevancia oportuna. Se puede considerar que es entonces cuando se acuña el término. Además, en esta contribución trazan un primer marco genérico de referencia para soportar la generación de IUs plásticas y una agenda de investigación para abordar su diseño y desarrollo teniendo en cuenta los resultados preliminares y el estado del arte en IPO.

Para incorporar la usabilidad y la necesidad de contemplarla, estos autores hacen uso de un símil, presentando la *plasticidad* como una característica de las IUs, inspirada en la propiedad de los materiales para expandirse y contraerse sin romperse bajo restricciones naturales, y por tanto preservando la integridad y un uso continuado de los mismos [TC99].

Trasladando esta propiedad al campo de la IPO, definen la ***plasticidad de la IU*** como:

Plasticidad 1 (Thevenin et al., 99) *La capacidad de un sistema interactivo de soportar variaciones en las características físicas tanto de la plataforma como del entorno, adaptándose a esos cambios contextuales preservando al mismo tiempo la usabilidad.*

Siguiendo con el símil de la propiedad de *"plasticidad de los materiales"*[3], la virtud que éstos tienen para expandirse y contraerse corresponde a la capacidad de adaptación de la IU bajo diversas condiciones fluctuantes del contexto de uso. Por otro lado, la *integridad* de los materiales guarda analogía con la propiedad de usabilidad, de manera que, así como los materiales se contraen y expanden sin llegar a romperse, por analogía las *IUs plásticas* son capaces de adaptarse a distintos factores contextuales y a distintas plataformas sin que por ello se degrade el nivel de usabilidad de las mismas. La usabilidad es, por tanto, la propiedad que debe preservarse para poder afirmar que una IU es plástica, distinguiéndola de una IU meramente adaptable.

Otra prueba de la relevancia que toma la usabilidad a partir de ese momento es que el lema que fundamenta el concepto de *plasticidad* incluye ya una referencia explícita de este nuevo requisito. Se enuncia así: "especifica una IU ***usable***, genera múltiples" [TC99].

Además, cabe señalar que en esta contribución se pone de manifiesto la preocupación por el coste de desarrollo de las distintas IUs a generar, así como por la consistencia entre las mismas. La adaptación debe ofrecerse sin incurrir en los elevados costes y esfuerzos de desarrollo que supondría tener que llevar a cabo un completo rediseño y re-implementación del sistema para cada plataforma. Como consecuencia de ello se establece un vínculo indisociable entre el concepto de *plasticidad* y una cierta facilidad de adaptación o amoldamiento

[3]Capacidad de un material de ser modelado.

a diversas variaciones, con el propósito de eludir ese sobreesfuerzo. Esta toma de consciencia abre el camino a la generación automática o semi-automática de IUs plásticas. Sin embargo, no es hasta el 2002 en [CCT+02a] que se hace explícito este requisito de minimización del coste de desarrollo en la propia definición de *plasticidad*.

Siguiendo con esta evolución cronológica, el concepto de plasticidad toma nueva forma en el 2000 por parte del mismo grupo IIHM. La diferencia es que en la propia definición se integra el término *contexto de uso*, obviando qué componentes lo forman [CCT00]. De este modo se consigue generalizar y formalizar el concepto de plasticidad, evitando que las posibles interpretaciones del *contexto de uso* puedan afectar en la concepción del mismo. No obstante, es ineludible que para conocer el ámbito de aplicación de la *plasticidad* de un sistema interactivo particular deba concretarse qué tipo de consideraciones intervienen en la caracterización del *contexto de uso*.

La nueva definición aportada es:

Plasticidad 2 (Calvary et al., 00) *La capacidad de un sistema interactivo de soportar variaciones en el contexto de uso mientras se preserva la usabilidad.*

El siguiente paso se produce en [CCT01b]. Aparece una definición donde se hace alusión explícita a que el efecto de los cambios contextuales sobre el sistema interactivo debe reflejarse sobre la IU del mismo. Hasta ese momento únicamente se había hablado de *"la capacidad de soportar esas variaciones"*, sin concretar la manera como materializar esos cambios. Se confirma con ello la *plasticidad* como una propiedad propia de la IU, hablando por tanto de *plasticidad de la IU*, haciéndose un hueco en el campo de IPO. Definen la *plasticidad* como:

Plasticidad 3 (Calvary et al., 01) *La habilidad de un sistema de moldear su propia IU a un rango de contextos de uso.*

Aunque la definición completa es la que aparece de *IU plástica* en [CCT01a]:

IU Plástica 1 (Calvary et al., 01) *IU capaz de proporcionar adaptación a distintos contextos de uso mientras preserva un conjunto predefinido de propiedades de usabilidad.*

A pesar de ello, posteriormente, en 2005, estos mismos autores abogan que no es tan sólo la IU la que puede verse afectada por los cambios en el contexto de uso, como se introduce más adelante.

Siguiendo este desarrollo cronológico acerca de la evolución del término, cabe destacar el trabajo llevado a cabo en el proyecto Europeo CAMELEON (Context Aware Modelling for Enabling and Leveraging Effective interactiON)[4], fundado en 2002, cuyo objetivo principal es el de soportar el diseño y desarrollo de sistemas software interactivos plásticos. Se trata de un proyecto que reúne la experiencia probada de diversas técnicas y herramientas desarrolladas por grupos destacados en el campo de la plasticidad, entre ellos el grupo IIHM. En el 2002 se publica un informe donde se describen los modelos, técnicas, herramientas y métodos desarrollados hasta el momento para la generación de IUs plásticas, los cuales se integran en un marco general de desarrollo de IUs plásticas: el CAMELEON Reference Framework [CCT+02a].

La definición de *plasticidad de la IU* que se aporta en este documento es la siguiente:

Plasticidad 4 (Calvary et al., 02) *La capacidad de adaptar una misma IU genérica a múltiples contextos de uso, soportando variaciones en diversas circunstancias contextuales sin descuidar la usabilidad, y al mismo tiempo minimizando los costes de desarrollo y mantenimiento.*

Se trata de la definición más completa hasta el momento sobre el término, cuyo aporte es la incorporación del concepto *IU genérica*, que pasa a ser el objeto que recibe la acción de adaptarse, en lugar de ser la IU como en *Plasticidad 3*, o el sistema interactivo como en *Plasticidad 1* o *Plasticidad 2*. Se define una IU genérica como:

IU genérica (Calvary et al., 02) *Especificación abstracta, genérica, independiente de dispositivo y única, suficientemente flexible como para afrontar múltiples fuentes de variación, y a partir de ella producir tantas IUs como sean necesarias.*

Se trata de una descripción de la disposición y composición de la IU (expresada como un conjunto de objetos de interacción abstractos), de su estructura estática y de su evolución en el tiempo. Esta especificación debe expresarse a través de un lenguaje de modelado.

En cualquier caso, la meta a conseguir es la misma: una *IU plástica* es la que especificándose una sola vez es capaz de derivar en distintas IUs apropiadas para distintas plataformas y variaciones en el entorno, preservando la usabilidad y procurando de este modo minimizar el coste de desarrollo y mantenimiento, como ya se vislumbraba en los lemas con los que se acuñó el término, introducidos anteriormente.

En 2004 se realiza una profunda revisión del concepto de plasticidad [CCD+04], identificando varias razones para un nuevo replanteamiento, las cuales están basadas en la

[4] http://giove.cnuce.cnr.it/cameleon.html (IST-R&D : Plasticity of User Interfaces ; 2001-2004).

experiencia. Entre otras cosas, la consideración de que la funcionalidad del sistema interactivo también puede verse afectada por los cambios en el contexto de uso, como se ha comentado anteriormente. Este hecho puede visualizarse con el siguiente escenario: un usuario que utiliza una PDA se desplaza a una nueva ubicación en la que hay un nuevo servicio disponible; entonces ese nuevo servicio aparece en su PDA como servicio apto para ser utilizado. En este caso se ha producido no sólo un cambio en la apariencia de la IU, sino que también se le está ofreciendo una nueva funcionalidad al usuario. Basándose en esta premisa, proponen enfocar la aplicación de la plasticidad a un nivel de granularidad mucho más fino: el elemento de interacción, en lugar de hacerlo a nivel de sistema interactivo, como hasta entonces. Desaparece, por tanto, cualquier alusión a la IU en la definición de *plasticidad.* En particular, el autor de esta tesis está de acuerdo con esta última consideración, defendiendo que el efecto de la adaptación no tiene porqué estar limitado a los componentes de la IU.

Pero quizás el aspecto más relevante de la revisión realizada en [CCD+04] es que la referencia explícita de la "*preservación de la usabilidad"* en la definición relega a un segundo plano el aspecto de utilidad o funcionalidad del sistema interactivo, descuidando su importancia. De hecho, en 2003 los revisores del proyecto Europeo CAMELEON [CCT+02a] criticaron que la definición no era suficientemente operacional. Con el objetivo de ofrecer la posibilidad de especificar también los requisitos concernientes a la preservación de las propiedades funcionales (por ejemplo, la consecución de las tareas), el ámbito de la definición debía ser extendido. Esta expansión se materializa mediante el reemplazo del término usabilidad por el de *calidad en uso* (ISO 9126-4), refiriéndose éste a la definición ISO [ISO03a]. Como es enunciado por ISO, la *calidad en uso* está basada en propiedades internas y externas que incluyen eficiencia, fiabilidad, etc., y entre la que se incluye la usabilidad. De acuerdo a estas normas, la definición es ahora reforzada por un conjunto de características (factores), sub-características (criterios) y métricas [ISO03b]. Calvary et al. defienden que la adaptación no debe limitarse al nivel de presentación, sino que debe llegar al núcleo funcional. La nueva definición de *plasticidad* presentada en [CCD+04] es la siguiente:

Plasticidad 5 (Calvary et al., 04) *La habilidad de un sistema interactivo de soportar variaciones en el contexto de uso mientras se preserva la calidad en uso.*

Paralelamente, la de *IU plástica* es la siguiente:

IU Plástica 2 (Calvary et al., 04) *IU capaz de proporcionar adaptación a distintos contextos de uso mientras preserva un conjunto predefinido de propiedades de calidad en uso, entre las que se incluyen las de usabilidad.*

Se insiste en el hecho de que la plasticidad no es una propiedad absoluta, sino que es relativa a un conjunto de elementos, los cuales ya han sido mencionados anteriormente. Según se expresa en [CCD+04]

> *"un sistema interactivo es considerado plástico para un conjunto de propiedades y de contextos de uso concretos si es capaz de garantizar esas propiedades mientras se lleva a cabo la adaptación".*

Ahora ese conjunto de propiedades no se limita a unos atributos de usabilidad, sino que debe ser seleccionado y posteriormente evaluado en base a las normas ISO mencionadas. Por ejemplo, la latencia y estabilidad del sistema interactivo con respecto a la "eficiencia", como una de las características de calidad.

Por último, en esta contribución se le da un nuevo enfoque a la meta de la plasticidad, presentándola más bien como un *área de investigación* que surge para describir, catalogar y estudiar la problemática derivada de la diversidad de contextos de uso, con el fin de proporcionar soluciones de manera económica y ergonómica.

2.1.2. Aspectos inherentemente relacionados al concepto de plasticidad

Retomando esta última definición de *IU plástica*:

IU Plástica 2 (Calvary et al., 04) *IU capaz de proporcionar* ***adaptación*** *a distintos* ***contextos de uso*** *mientras preserva un conjunto predefinido de propiedades de* ***calidad en uso****, entre las que se incluyen las de* ***usabilidad****.*

aparecen varios términos que requieren igualmente ser definidos para llegar a una completa comprensión del término. Se trata de los conceptos de (1) *contexto de uso,* (2) *adaptación* y (3) *preservación de la calidad en uso o usabilidad.*

(1) No existe una definición consensuada acerca de lo que es el **contexto de uso** y lo que éste comprende. En un amplio sentido, se puede entender el *contexto* como

> *"las condiciones interrelacionadas en las cuales un evento, acción o situación tiene lugar"*[5]

donde se hace alusión a que los elementos que intervienen en el contexto están interrelacionados y guardan una relación de coherencia, otorgando un cierto significado a la acción, evento o situación que se está caracterizando.

La definición de **contexto de uso** que se estima más conveniente en la presente tesis es la siguiente:

[5]Extracto de Merriam Webster. http://www.m-w.com

Definición 2.1 (Contexto de uso): Confluencia de valores de variables que caracterizan el conjunto establecido de atributos o parámetros que el sistema se compromete a tomar en consideración, y que conducen a un entendimiento más o menos profundo y sofisticado de la situación en la que se encuentra el usuario a lo largo de la interacción con el sistema. □

En otras palabras, se entiende como cualquier tipo de información útil para describir la situación particular en la que un determinado usuario se encuentra realizando una tarea concreta soportada por un sistema interactivo. Dicha información suele describirse a través de un conjunto de variables o parámetros.

Generalmente este conjunto de variables agrupa el contexto operativo –descripción de los recursos del sistema computacional-, el contexto físico donde la interacción toma lugar -condiciones del entorno-, y el perfil del usuario -conjunto de características, necesidades y preferencias que lo describen-. No obstante, a pesar de que se han llevado a cabo muchos trabajos con el fin de describir el *contexto*, no existe un consenso ni un estándar para identificar tipos generales de contexto y establecer cuál es el conjunto adecuado de parámetros a considerar, que suele dejarse de manera ad hoc, y en función del dominio de aplicación [DA00b]. Así, algunas de las contribuciones más relevantes al respecto son las siguientes:

- La de Schilit, que divide el contexto en tres categorías: el *contexto computacional*, el *contexto del usuario* y el *contexto físico* [SAW94].
- La de Chen y Kotz [CK00], que añaden una cuarta categoría de contexto: el *tiempo*, distinguiendo así un total de cuatro componentes: físico, computacional, de tiempo y el contexto de usuario.
- La de Dey y Abowd en [DA00b], que fijan estos cuatro tipos de contexto: localización, identidad, actividad y tiempo. En particular, la actividad responde a cuestiones fundamentales sobre qué está ocurriendo en una situación concreta.
- Otros intentos realizados con el fin de identificar diferentes tipos de contexto son los de Brézillon en [BP99b] y [BBPP04]. Se distinguen tres partes principales de contexto, las cuales pueden ser entendidas como tres ámbitos o rangos distintos: el *conocimiento externo*, el *conocimiento contextual* y el *contexto procedimental*. Por otro lado, en [BBPP04] distinguen entre contexto de grupo, de proyecto y el contexto individual.

En cualquier caso, la filosofía subyacente es la misma: el contexto es utilizado para interpretar las necesidades e intenciones del usuario, y adaptar la aplicación y su interfaz

de acuerdo a las mismas [CK00], [SAW94]. Puede verse una revisión detallada de estos términos en [AM00], [CK00], [KRK+02].

Como cabía suponer, la caracterización del *contexto de uso*, y por tanto, la decisión del conjunto de atributos a tener en cuenta para el estudio de la plasticidad ha sufrido también variaciones a medida que la noción de plasticidad tomaba forma. Así, según Calvary, Coutaz y Thevenin en [CCT00], el *contexto de uso* para un sistema plástico cubre dos clases de atributos: la plataforma hardware y software utilizada para interactuar con el sistema y las características físicas del entorno donde se lleva a cabo la interacción, constituyendo una *tupla*[6] de dos elementos. De hecho, es así como se acuña el término en *Plasticidad 1* (véase *sección 2.1.1*), haciendo mención de esos dos atributos. No es hasta 2003 que se introduce un atributo más: el usuario que hace uso del sistema interactivo [CCT+03]. Se hace mención explícita del uso de esta tupla (usuario, plataforma, entorno) en [DC03].

En particular, el conjunto de atributos que se establecen en esta tesis para describir el *contexto de uso* se introducen en el siguiente apartado (véase *sección 2.1.3*).

(2) Se entiende por **adaptación** la reacción apropiada del sistema ante un cambio en el contexto de uso del mismo [CCT01a]. Atendiendo a diversos factores, la adaptación puede catalogarse de diversas maneras:

2.1) En función de la naturaleza y envergadura del cambio (*simples/complejas*). La adaptación puede consistir en un ajuste puntual del sistema interactivo al nuevo contexto de uso, en busca de una mejor armonización conjunta, o bien puede comportar un efecto mucho más global y notable que puede implicar incluso la sustitución de la IU actual por otra distinta, o como mínimo una remodelación de la misma. Podemos hablar de adaptaciones *simples* o *complejas*. Además, como ya se introdujo en [CCD+04], también puede tener un impacto en la funcionalidad del sistema interactivo. En general, en estos casos se hablaría de una adaptación *compleja*.

2.2) En función de la intervención del usuario en la especificación de su perfil (*adaptable/adaptativa*). En el campo de IPO, y en particular en el de la *personalización* [Bru96] (adaptación a un usuario en particular), es ampliamente reconocida la distinción entre dos propiedades complementarias de la adaptación: la *adaptabilidad* y la *adaptatividad* [BNR90], [IFI97]. *Adaptabilidad* es la capacidad de un sistema para adaptar un conjunto predefinido de parámetros [TC99]. En este caso es el propio usuario el que se da a conocer al sistema, especificando sus

[6] Secuencia ordenada y finita de objetos.

preferencias y cualidades, al tiempo que el sistema se basa en esos parámetros para llevar a cabo la adaptación. En este caso se habla de *sistemas interactivos o IUs adaptables*[7]. *Adaptatividad* es la capacidad del sistema para llevar a cabo una adaptación de manera automática, esto es, sin la intervención explícita por parte del usuario [TC99]. En este caso no es el usuario quien manifiesta sus cualidades y preferencias, sino que es el propio sistema el que de manera autónoma y en segundo plano lleva a cabo una tarea de monitorización para inferir las peculiaridades de cada usuario. La adaptación se llevará a cabo una vez inferidos dichos parámetros. En este caso se habla de *sistemas interactivos o IUs adaptativas*[8].

2.3) En función de la intervención del usuario en la aplicación de la adaptación (*automática/manual*). Independientemente de que la IU sea *adaptable* o *adaptativa*, podemos distinguir si la acción propia de la adaptación se lleva a cabo de manera autónoma y automática por parte del sistema, o bien es el usuario quien debe hacer explícita la necesidad de adaptación, interviniendo de algún modo. En este último caso se trata de una adaptación *manual*. En [TC99] ya se menciona esta distinción, aunque de manera superficial.

A pesar de la aparente similitud entre estos dos últimos criterios de clasificación (2.2 y 2.3), en el caso de la 2.2 lo que es susceptible de automatizar es la determinación del perfil del usuario. En cambio, en el caso de la 2.3 lo automatizable es la ejecución de la acción propia de la adaptación.

2.4) En función del nivel de operación o fase (diseño o ejecución) involucrada (*dinámica/estática*). Tal y como se menciona también en [TC99], la adaptación puede ser *dinámica* y *estática*. La adaptación *dinámica* es la que se efectúa en tiempo de ejecución, generalmente ofreciendo una respuesta en tiempo real. Su ejecución es por tanto inmediata, procurando ofrecer una cierta *anticipación a los cambios contextuales* (concepto que suele referirse como *proactividad*). La adaptación *estática* implica un rediseño o re-implementación de la IU, no resoluble en tiempo real.

En general, una adaptación *dinámica* se corresponde con una adaptación *simple*, de efecto puntual sobre la IU, mientras que una adaptación *estática* se corresponde con una adaptación *compleja*. No obstante, no siempre existe esta correspondencia. Así, si tomamos como ejemplo un sistema de soporte a la visita de un yacimiento

[7] Capaz de adaptarse, gracias a la intervención del usuario que es quien establece los parámetros de adaptación.

[8] Término inglés cuyo equivalente al español sería *adaptativo*. Según Dix [DFAB03] *adaptivity* es "la modificabilidad de la IU por parte del sistema".

arqueológico, el hecho de que el usuario cambie su orientación, y como consecuencia, varíe el punto de interés o elemento a estudiar del propio yacimiento, puede ocasionar una remodelación de la IU con el fin de mostrar la información referente a ese nuevo elemento. El que dicha remodelación pueda ser resuelta en tiempo real o requiera de una fase de rediseño depende de muy diversos factores, entre los que entraría en juego la capacidad de cómputo y de conexión en red del dispositivo utilizado. No es lo mismo, por ejemplo, trabajar con un teléfono móvil que con un Tablet PC. Si se resuelve en tiempo real se trataría de una adaptación *compleja, automática* y *dinámica*. En caso contrario se trataría de una adaptación igualmente *compleja* y*automática, pero estática.*

Estas distinciones juegan un papel esencial en el desarrollo de la presente tesis. Como se expone en detalle más adelante en este mismo capítulo, distinguimos entre *plasticidad explícita* y *plasticidad implícita*. La *plasticidad explícita* corresponde a una adaptación *estática y compleja* (desencadenada de forma manual o *automática*), y la *plasticidad implícita* a una adaptación *automática* y *dinámica* (ya sea o no *compleja*, como en el caso del ejemplo anterior). Un ejemplo de *plasticidad implícita simple* se daría cuando el propio sistema regula automáticamente el contraste de la pantalla en función de la luminosidad externa. Por su parte, un ejemplo de plasticidad explícita para el caso del ejemplo manejado sería que cuando cambia el punto de interés se remodele la IU modificando la información a mostrar (compleja), siendo necesario recurrir a una fase de rediseño para su obtención(estática). Esa necesidad puede ser explícitamente requerida por el usuario (manual), o bien el propio sistema podría detectar esa necesidad (automática), por ejemplo haciendo un seguimiento de la localización del usuario a través de un dispositivo GPS.

(3) Una IU que soporta adaptación **preserva la calidad de uso** si las propiedades seleccionadas en la fase de diseño para medir esa calidad de uso de un sistema interactivo particular se mantienen dentro de un rango predefinido de valores, a medida que la adaptación a los cambios contextuales se va produciendo [CCD+04]. Se destaca de este modo la necesidad de especificar y evaluar esas propiedades en base a las características y sub-características proporcionada por ISO [ISO03a].

Del mismo modo, una IU que soporta adaptación **preserva la usabilidad** si las propiedades seleccionadas en la fase de diseño para medir la usabilidad de un sistema interactivo particular se mantienen dentro de un rango predefinido de valores, a medida que la adaptación a los cambios contextuales se va produciendo [CCT00]. En otras palabras, la adaptación debe llevarse a cabo sin degradar el grado de usabilidad. Precisamente, y tal y como ya se ha expuesto anteriormente, este es el sello que distingue la plasticidad de una mera adaptación.

2.1.3. Caracterización del *contexto de uso* en la presente tesis

Para el propósito del estudio llevado a cabo en esta tesis, y concretamente para caracterizar las IUs plásticas, el *contexto de uso* se describe en función de cuatro atributos, los cuales forman una tupla de cuatro elementos: (usuario, plataforma, entorno, tarea en curso), a tener en cuenta en entornos de trabajo individuales.

No obstante, en entornos colaborativos o de trabajo en grupo se propone incorporar una quinta componente. Se trata de la ***consciencia de grupo***. Por consiguiente, con el fin de proporcionar plasticidad a los sistemas colaborativos, los atributos que se establecen en esta tesis para describir el contexto de uso son cinco, constituyendo en este caso tuplas de cinco elementos: (usuario, plataforma, entorno, consciencia de grupo, tarea en curso).

1) **El usuario** particular que se encuentra realizando una tarea concreta del sistema.

Los usuarios potenciales del sistema representan estereotipos, que de acuerdo al *Human Processor Model* [CMN83] pueden describirse como un conjunto de valores que caracterizan las capacidades y condiciones cognitivas, perceptivas o procedimentales de aquéllos: los llamados *perfiles de usuario*. El manejo de perfiles de usuario establecidos de antemano por parte de un sistema interactivo se corresponde con el concepto de *sistema* o *IU adaptable*, introducido en la *sección 2.1.2*.

Es importante mencionar que en muchos casos se maneja otro tipo de información adicional a efectos de describir aquellos aspectos de naturaleza dinámica cuyo conocimiento pueda ser también de utilidad para comprender la vivencia particular que un usuario está experimentando con el sistema en un momento determinado. Algunos ejemplos son: preferencias detectadas durante el proceso de interacción, necesidades puntuales en la realización de una tarea, estado cognitivo, patrones de comportamiento y actitudes propias de un individuo en particular, etc., las cuales incluso pueden ser cambiantes a medida que se avanza en el uso del sistema.

Con este tipo de aspectos se trata de tomar consciencia de las peculiaridades propias de cada individuo, con el fin de configurar un perfil individualizado. Este perfil únicamente puede ser deducido de una monitorización en tiempo real y de una inferencia subyacente de los patrones de actuación llevados a cabo por el usuario durante la fase de ejecución del sistema. Este proceso corresponde al de un *sistema interactivo adaptativo*, introducido en la *sección 2.1.2*.

Este tratamiento puede ser combinado con el manejo de información proporcionada por un perfil de usuario, dando lugar a un *sistema o IU adaptable y adaptativo*.

2) La plataforma hardware y software que da soporte al sistema interactivo, a fin de llevar a cabo una tarea concreta.

Se trata de detallar todo tipo de información relativa al equipo de computación, dispositivos de interacción, entorno de red y plataforma software que configuran el entorno operativo en el que se va a llevar a cabo la interacción.

Este aspecto suele modelarse en términos de cuantificación y/o medición en tiempo real de los recursos hardware y software, los cuales determinan la forma en que la información es computarizada, transmitida, visualizada y manipulada por los usuarios, tal y como se propone en [TC99] a través de una heurística, y que se presenta en el *Capítulo 3* (véase *sección 3.3.1.2*; *Heurística 3*).

Debido al amplio abanico de variaciones en recursos hardware existente, se propone realizar la siguiente clasificación, que además resultará útil para posteriores alusiones en el presente documento.

Aspectos hardware que afectan directamente en la reconfiguración de la IU

Se trata de los aspectos más populares en el campo de la *plasticidad de la IU*. Son los siguientes: tamaño y resolución de pantalla, características de los dispositivos de entrada disponibles (dispositivo apuntador, pantalla táctil, teclado -en ocasiones alfanumérico-, botón de navegación de 4 ó 5 direcciones, botones hardware, teclas especiales de función y de software), así como otro tipo de potencialidades como son las capacidades gráficas, el número de colores soportados (o profundidad del color), potencial de sonido, etc.

Aspectos tecnológicos y de potencialidades hardware que afectan en el rendimiento y funcionalidad del sistema interactivo

Por lo que respecta a potencialidad hardware se consideran: la capacidad de procesamiento, protocolo de red, protocolo de comunicación (GPRS, UMTS, y sus respectivas versiones homólogas perfeccionadas EDGE[9] y HSPA[10]), conectividad inalámbrica (puerto de infrarrojos, Bluetooth, WiFi), etc. Otro aspecto importante es la disponibilidad o no de gran cantidad de funcionalidades adicionales, algunas de ellas exclusivas de los teléfonos móviles, como son: tipo de mensajería soportada

[9]Enhanced Data rates for GSM Evolution (EDGE) o también Enhanced GPRS (EGPRS), es una tecnología de telefonía móvil digital que mejora el ratio de transmisión de datos, así como la fiabilidad de transmisión de datos.

[10]High-Speed Packet Access (HSPA) es una colección de protocolos de telefonía móvil que extienden y mejoran el funcionamiento de los protocolos UMTS existentes.

(SMS[11], MMS[12], EMS[13], e-mail) en el caso de móviles, capacidad para manipular el sonido y el vídeo, y también numerosas extensiones periféricas tales como reproductores MP3, cámaras digitales, radio FM, entre otros, que en un momento dado pueden complementar la funcionalidad núcleo de este tipo de aplicaciones novedosas.

Aspectos tecnológicos de naturaleza dinámica

Se trata de tomar en consideración diversas restricciones técnicas de naturaleza dinámica, esto es, susceptibles de fluctuar durante la ejecución del sistema, las cuales pueden ser determinantes para una correcta consecución de las tareas. Algunas de ellas son: el ancho de banda, la autonomía de la batería, el estado de la red de conexión y del/de los servidores con los que se debe permanecer conectado, la intermitencia en la transmisión, la capacidad de memoria disponible -ésta es susceptible de variaciones significativas en dispositivos compactos-, etc. Es obvio que el manejo de este tipo de información dinámica adicional requiere algún mecanismo de detección subyacente, abriendo el camino a una posible *adaptación dinámica*.

En relación a la plataforma software, aún centrándonos en el caso de los dispositivos móviles, la variedad es tan o más inmensa que en el caso de los recursos hardware. Los aspectos software más destacados que deben ser diferenciados, y por tanto considerados en la meta de alcanzar una *independencia de dispositivo*[14] son el sistema operativo -en adelante S.O.-, y la versión de la máquina virtual de Java –en adelante JVM[15]-, teniendo en cuenta que los desarrollos presentados en este trabajo están enmarcados en el mundo de Java. La gran diferencia entre las plataformas Web y de sobremesa, con respecto a las plataformas móviles conlleva la necesidad de desarrollar no sólo diferentes versiones de un mismo S.O., sino también, en este caso, distintas ediciones de la plataforma 2 de Java, con sus respectivas JVMs, adaptadas a las restricciones y características de cada una. Así, la máquina virtual para J2SE (Java 2 Standard Edition)[16] es, por supuesto, mucho más potente que la de J2ME (Java 2 Micro Edition)[17], contando con un mayor número de APIs[18], notablemente más extensas y completas. Aún existiendo esta clasificación,

[11] Short Messaging Service
[12] Multimedia Messaging Service
[13] Enhanced Messaging Service. Una extensión a nivel de aplicación al servicio SMS para teléfonos móviles, a medio camino entre SMS y MMS, disponible en las redes GSM, entre otras.
[14] Capacidad de conmutar entre un conjunto predefinido de plataformas de computación, sin perder el contexto de la ejecución.
[15] del término inglés Java Virtual Machine.
[16] Edición orientada principalmente al desarrollo de aplicaciones para ordenadores de sobremesa.
[17] Edición de Java diseñada para dispositivos compactos, que utiliza implementaciones adaptadas a sus limitaciones.
[18] Application Programming Interface.

la diversidad en cada grupo es tremenda. Tan sólo en el mundo de las PDAs podemos encontrar un sinfín de JVMs, en ocasiones sin la correspondiente catalogación –esto es, sin la especificación del perfil y configuración, conceptos definidos a continuación.

Cabe añadir también la gran diferencia entre distintos S.O.s, no sólo entre la versión para equipos de sobremesa y sus homólogas versiones compactas, sino incluso en el propio mundo de los dispositivos móviles. En efecto, muchos de los teléfonos móviles funcionan sobre el S.O. Symbian[19], (el S.O. más extendido entre los SmartPhones[20]), mientras que las PDAs y organizadores corren sobre diferentes versiones del Palm OS, RIM OS[21], Windows Mobile para Pocket PC (los antiguos Microsoft Windows CE -o simplemente Pocket PC- y Microsoft SmartPhone), e incluso Linux, que se introdujo en este campo con una PDA diseñada por Sharp.

A pesar de que en el mundo de desarrollo Java se pretende acotar esta heterogeneidad entre dispositivos, definiendo conceptos como los de *configuración*[22] y *perfil*[23] -con los que se intenta componer agrupaciones o familias de dispositivos-, el hecho de que continuamente aparezcan en el mercado terminales móviles provistos de aplicaciones nativas y *APIs opcionales*[24], la mayoría propietarias[25] o incluso *APIs específicas de dispositivo*[26], dificulta aún más la posibilidad de consensuar una catalogación universal a nivel de desarrollo software. Esta notable dificultad, que da lugar a que los dispositivos se distingan adicionalmente por la funcionalidad específica con la que vienen provistos, ha sido calificada con el nombre de ***problema de fragmentación de APIs***[27] en [You05].

Distinguir entre las distintas configuraciones y perfiles a los que pertenece cada dispositivo, así como los muy variados paquetes y/o APIs opcionales con los que vienen provistos, resulta indispensable como primer paso para el desarrollo de *software a medida*[28]. Todo

[19] http://www.symbian.com

[20] Teléfono inteligente. Dispositivo de mano que integra la funcionalidad de un teléfono móvil, PDA o similar. Suele integrar funciones adicionales como comunicación a través de Wi-Fi, y el envío de e-mails.

[21] S.O. propietario desarrollado por Research In Motion, propio de los teléfonos BlackBerry, especializados en el manejo de correo electrónico inalámbrico.

[22] Es la parte de la plataforma software de un dispositivo móvil que define una JVM y un conjunto mínimo de bibliotecas de clases (APIs). La configuración proporciona la funcionalidad básica para un conjunto particular de dispositivos con características similares, o familias de dispositivos.

[23] Conjuntos de APIs de alto nivel que dan soporte a un entorno de ejecución para dispositivo móvil, en función de categorías específicas de los dispositivos. Los perfiles son particulares para configuraciones concretas y se combinan con ellas. Aquellas no pueden subsistir sin un perfil.

[24] Ofrecen un conjunto de bibliotecas de clases para soportar tecnologías nuevas y adicionales que son de uso específico, como por ejemplo Bluetooth, servicios Web o conectividad a bases de datos.

[25] APIs exclusivas de una marca.

[26] que en ocasiones causan que un conjunto muy reducido de dispositivos sea singular también a nivel de programación.

[27] Del término inglés API fragmentation problem.

[28] Se trata de desarrollar software para un conjunto muy específico de dispositivos, a veces pertenecientes a una misma marca propietaria. Este concepto se aproxima al de *tailoring* [Fer05].

ello dificulta en gran medida no sólo la simple portabilidad[29] de una misma aplicación entre distintos dispositivos, sino una meta mucho más ambiciosa, relativa al problema tratado en esta tesis. Se trata de la meta de plasticidad: la de ofrecer a cada sistema interactivo a desplegar en un dispositivo concreto aquellas características específicas que pueden ser explotadas de acuerdo a la funcionalidad propia con la que viene provisto ese equipo en particular. Un ejemplo sencillo sería que en una aplicación para un dispositivo móvil que contempla la posibilidad de utilizar el servicio de mensajería, se explotara esta funcionalidad según las posibilidades de cada dispositivo. Así, cuando esta aplicación se ejecutara en una PDA, esta extensión no se utilizaría, a no ser que la PDA estuviera provista de módem –por ejemplo, mediante conexión inalámbrica a un teléfono móvil-. En el caso de tratarse de un celular, sin embargo, esta funcionalidad podría ser explotada en mayor o menor grado con arreglo al tipo de servicio de mensajería propio del dispositivo, inclusive el correo electrónico en los modelos que lo soporten. En definitiva, el objetivo sería graduar las capacidades multimedia y de mensajería a las posibilidades del dispositivo.

En otros casos ese proceso de reconfiguración puede suponer una reducción significativa de la calidad (por ejemplo en la calidad gráfica), o de la cantidad de información a mostrar y en otras ocasiones un cambio importante en la operativa del sistema, como por ejemplo en el número de funcionalidades a ofrecer.

En definitiva, la meta de plasticidad desde el punto de vista de la *independencia de dispositivo* no se limita al objetivo de hacer funcionar una misma aplicación en distintos equipos, sino que se propone explotar al máximo la potencialidad hardware y software de la que se dispone, con el fin de obtener el máximo rendimiento y funcionalidad en cada dispositivo y para cada aplicación. Se trata de un moldeamiento o configuración a medida, que queda muy alejado del concepto de portabilidad, mucho más limitado. Así pues, la idea es tener en cuenta este tipo de consideraciones técnicas así como las capacidades de los muy variados recursos hardware y software que acompañan a este tipo de dispositivos, cuyo rango de potencialidad puede llegar a ser inmenso. En concreto, las variaciones hardware a las que se ha hecho referencia en esta última explicación son las que han quedado definidas en *Aspectos tecnológicos y de potencialidades hardware que afectan en el rendimiento y funcionalidad del sistema interactivo* (véase *sección 2.1.3*).

Todos los aspectos presentados aquí, en conjunto, constituyen factores determinantes a la hora de amoldar un determinado sistema interactivo a una determinada plataforma, a tener en cuenta en la caracterización del contexto de uso.

3) El entorno físico donde la interacción tiene lugar.

[29] Capacidad de un sistema de ejecutarse en múltiples plataformas, cubriendo los cambios en los recursos hardware y software, generalmente pertenecientes a una misma familia de equipos.

Para describir este atributo del contexto de uso convendría responder a la siguiente pregunta: ¿Cuáles son las características relevantes del entorno potencial en el que se piensan desarrollar las tareas del sistema? Según Coutaz y Rey en [CR02] el entorno denota

> *"el conjunto de entidades (objetos, personas y eventos) que rodean la actividad que se está llevando a cabo, las cuales pueden tener un impacto en el sistema y/o comportamiento de los usuarios, ya sea inmediato o futuro".*

De acuerdo a esta definición, el entorno puede abarcar el mundo entero. En general, la concepción acerca del *contexto* puede variar considerablemente, comprendiendo campos muy diversos. Han sido muchos los trabajos llevados a cabo con objeto de describir el *contexto*, tal y como se han presentado en la sección anterior.

En la práctica, serán los especialistas del dominio o campo de aplicación quienes establezcan los límites, identificando las entidades que son relevantes para cada caso concreto. De cualquier modo, la combinación de un conjunto rico de propiedades contextuales nos va a permitir alcanzar un entendimiento más preciso de cada circunstancia o contexto de uso. Así, por ejemplo, las condiciones de luminosidad constituyen un aspecto importante en la utilización de sistemas al aire libre, como por ejemplo un sistema de rastreo basado en visión por ordenador [CCRR02], o bien un sistema de soporte en la visita a un yacimiento arqueológico [YL04]. Las condiciones de sonoridad también pueden condicionar la utilización de sistemas interactivos. Así, por ejemplo, si se hace uso de una interfaz por voz, en el momento en que el entorno físico llega a un cierto grado de sonoridad, sería deseable que el sistema reaccionara anulando la transmisión sonora y supliéndola por una interfaz textual. Es indudable que la localización en el espacio proporciona información relevante acerca del entorno en el uso de sistemas móviles. De acuerdo a estos factores, tareas que pueden ser principales en una ubicación, como por ejemplo la de escribir en un papel en una oficina, se convierten en secundarias en situaciones distintas, como por ejemplo en un trayecto en tren.

Estos ejemplos hacen alusión a una cierta *"capacidad para personalizar los sistemas a las condiciones físicas del entorno"*. Esta capacidad es la que caracteriza los *sistemas sensibles al contexto*[30], y en concreto la *Computación Sensible al Contexto* [DFAB03], [DA00b], [CK00]. Estos sistemas acostumbran a soportar tareas que requieren cierta movilidad. Por consiguiente, las condiciones físicas que caracterizan el entorno donde tareas se llevan a cabo tienden a ser dinámicas y cambiantes, como es el caso de la localización, del nivel de sonoridad y de la luz ambiental mencionados anteriormente. Ese dinamismo contribuye a

[30] Un sistema es sensible al contexto si usa el contexto para proporcionar información relevante y/o servicios al usuario, donde dicha relevancia depende de la tarea del usuario [DA00b].

la necesidad de una detección y reacción inmediatas (en tiempo real) por parte del sistema ante un cambio en las restricciones del entorno. Estas son las características que debe cumplir un sistema para que pueda considerarse *sensible al contexto*. Para ser más precisos, este tipo de sistemas deben estar provistos de (1) un sistema de monitorización de la información acerca del entorno y de detección de cambios; (2) un mecanismo de recogida de la información relevante y de decisión acerca de la reacción a tomar por parte del sistema; y finalmente (3) un soporte de ejecución de esa reacción, responsable de llevar a cabo la adaptación.

El campo de la *Computación Sensible al Contexto*, que tiene más de una década de antigüedad, ha centrado la mayor parte de sus esfuerzos en el estudio y tratamiento de la localización para aplicaciones móviles, reduciendo su potencialidad y problemática exclusivamente a una *Sensibilidad a la Localización*[31], sub-área de la *Computación Sensible al Contexto* ampliamente reconocida [DFAB03]. En particular, desde el lanzamiento de los llamados Servicios Basados en Localización o LBS[32] (del inglés *Location-Based Services*) a finales de los años 90, se han llevado a cabo numerosos trabajos de investigación, prototipos y sistemas tratando de dar aplicación y explotar esta potencialidad. La localización es actualmente una de las modalidades más potentes y extendidas de personalización de los servicios móviles.

4) **El *awareness***[33], comúnmente traducido como *consciencia de grupo*.

Es el término utilizado en la comunidad CSCW[34] para referirse al entendimiento compartido o conocimiento de la actividad desarrollada en un grupo de trabajo, tal y como se ha definido en el *Capítulo 1* (véase *Capítulo 1 sección 1.2*. Incluye cualquier conocimiento derivado del desempeño de un objetivo común, a fin de facilitar un mayor entendimiento de la situación en la que se encuentra el grupo, y de ese modo contribuir a una mayor efectividad en la actividad grupal.

Entre los factores que lo componen se encuentra el conocimiento de qué actividades han sido completadas por el grupo de trabajo, cuáles se están llevando a cabo por parte de los miembros integrantes, el estado del grupo, las condiciones y restricciones particulares en las que se está desarrollando la actividad grupal, otras características específicas del grupo de trabajo, como por ejemplo su contexto organizacional, etc. De igual forma incluye aspectos como quiénes están trabajando, dónde lo están haciendo, qué artefactos están

[31] Traducción del término inglés *Location-Awareness*.

[32] Servicio consitente en el envío de publicidad personalizada y otro tipo de información basada en la localización actual del usuario de un teléfono móvil que se haya subscripto a dicho servicio.

[33] Entendimiento de las actividades de los otros integrantes de un grupo de trabajo, que proporciona un contexto para la propia actividad [DB92].

[34] Computer Supported Cooperative Work.

manipulando, el por qué de algunas cuestiones y decisiones, etc. En resumen, todo tipo de información que puede irse generando en el transcurso de la actividad grupal. Como puede observarse, algunos de estos factores son de naturaleza dinámica (como por ejemplo el estado del grupo), mientras que otros permanecen fijos a lo largo del desarrollo (como son las características específicas del grupo).

Este aspecto constituye una componente adicional, a incorporar en el caso entornos de trabajo colaborativo soportado por computador, con el propósito de fomentar una colaboración real entre los miembros integrantes del grupo de trabajo. Puede considerarse como la componente relativa al *contexto social*[35], un tipo de contexto particular para aplicaciones colaborativas. De hecho, la necesidad de considerar aspectos de *consciencia de grupo* como parte de la información contextual en sistemas colaborativos es ampliamente reconocida en el campo de *groupware*[36]. Así, por ejemplo, Gutwin y Greenberg reconocen que proporcionar este tipo de información a los miembros del grupo facilita la comprensión no sólo de cómo sus acciones encajan en las metas de grupo, sino también de cómo las acciones de sus compañeros afectan en las primeras. Disponiendo de este entendimiento, los miembros del grupo se sienten más capaces para elegir la respuesta apropiada a un conjunto de posibilidades [GG02].

Así, esta componente del contexto de uso presenta una doble perspectiva: (1) la perspectiva individual que tiene cada miembro acerca del grupo; y (2) una perspectiva global, fruto de la confluencia y consideración conjunta de todas y cada una de las perspectivas individuales. En efecto, si lo que se pretende es obtener una colaboración real, es esencial que cada miembro comparta su visión particular con el resto del grupo, a fin de enriquecer la información y la percepción grupal.

Por consiguiente, en la presente tesis se distinguirán estos dos conocimientos relativos al grupo, introduciendo dos tipos específicos de *awareness*: la *consciencia de grupo particular*, y el así conocido *shared-knowlege awareness*, que suele traducirse como *Consciencia de conocimiento compartido*, los cuales definimos a continuación.

Se define la *Consciencia de grupo particular* como:

Consciencia de grupo particular (Dourish et al., 92) *Percepción o entendimiento que cualquier miembro de un grupo de trabajo tiene de las actividades desarrolladas por los demás miembros desde una perspectiva individual, el cual proporciona un contexto para la*

[35]Todo aquello relativo a una agrupación natural o pactada de personas que cooperan mutuamente en un fin común [AGOP05].

[36]Campo de investigación que estudia los sistemas basados en computador que soportan grupos de personas comprometidas en una misma tarea (o meta) y que proporcionan una interfaz para un entorno compartido [EGR91].

propia actividad que deberá ser considerado por el propio miembro del grupo con el objetivo de llevar a cabo la actividad grupal de manera colaborativa.

Este tipo de *awareness* proyecta el estado y los cambios del entorno compartido hacia los usuarios, proporcionando información contextual del tipo dónde, cuándo, qué, quién, cómo, con el propósito de que los usuarios entiendan cómo sus acciones contribuyen a la consecución de las metas del grupo y puedan regular su comportamiento [DB92], [FKL+98].

Para definir la *Consciencia de conocimiento compartido* conviene definir previamente el *conocimiento compartido.*

Conocimiento compartido (Collazos et al., 02) *Conocimiento que va más allá de un entendimiento compartido del problema y que abarca todos aquellos aspectos del trabajo colaborativo que puedan ser de utilidad en el desarrollo de la actividad grupal, desde una perspectiva global. Comprende información sobre las restricciones del grupo, las actividades de los miembros integrantes y cualquier otro detalle acerca de qué, cómo, quién, dónde se está realizando, etc., así como información relativa al conocimiento de los miembros del grupo (en ocasiones incluso a lo que se está aprendiendo), proporcionando una visión de conjunto.*

Esta visión de conjunto, que se compone del entendimiento de varios aspectos del trabajo colaborativo, así como la recopilación de cada una de las percepciones particulares de cada miembro, permite alcanzar un mejor entendimiento de la actividad grupal, con el propósito de fomentar una colaboración real en escenarios colaborativos distribuidos [CGPO02]. Se trata de una co-construcción que requiere no sólo la contribución de todos los participantes, sino también su aceptación [CS89], dado que para que sea útil deberá ser accesible de algún modo por los mismos. Según [CGPO02], donde se introduce este término en el contexto de un escenario de *aprendizaje colaborativo*[37],

> *"cuanto mayor es el conocimiento compartido, más efectivo puede ser el aprendizaje".*

En consecuencia, se define la *Consciencia de conocimiento compartido* como:

Consciencia de conocimiento compartido (Collazos et al., 02) *Entendimiento que un grupo de personas tiene acerca de su conocimiento compartido, el cual proporciona un contexto para la actividad grupal que deberá ser asimilado por todos los miembros del grupo con el objetivo de llevar a cabo dicha actividad de manera colaborativa.*

Para poder disponer de este entendimiento global, se requiere (1) algún tipo de soporte donde mantener ese *conocimiento compartido*, ya sea tipo repositorio común o memoria

[37] del término inglés Computer Supported Collaborative Learning (CSCL).

de grupo de trabajo; y (2) algún mecanismo que facilite no sólo la recopilación de las percepciones individuales de cada uno de los miembros del grupo, sino también su posterior distribución a cada uno de ellos. Dourish et al. mantienen que esa *consciencia del conocimiento compartido* debe ser recogida de manera pasiva (implícita), para ser distribuida al resto de los miembros del grupo también implícitamente [DB92]. Aquí es donde intervienen los *Mecanismos de awareness*.

Es ampliamente reconocido que para disponer de este tipo de información relativa a la *consciencia de grupo* se requieren mecanismos específicos para su captura y construcción. Son los llamados *mecanismos de awareness*, que serán referidos en lo sucesivo como *mecanismos de consciencia de grupo* [CGPO02].

Mecanismos de consciencia de grupo (Collazos et al., 02 - Alarcón et al., 04) *Cualquier mecanismo de soporte en la construcción y mantenimiento del conocimiento relativo a la actividad grupal que contribuya a un mayor entendimiento y promueva la interacción entre los miembros del grupo, con el fin de proporcionar una mayor efectividad en el trabajo colaborativo* [CGPO02].

Ejemplos: diferentes mecanismos de interacción entre los miembros, como son los mecanismos de notificación, técnicas de percepción, soporte para la memoria de grupo, conocimiento compartido, política de privacidad, estrategias de coordinación, comunicación, monitoreo, etc. [AGOP05].

Estos mecanismos constituyen un aspecto vital en entornos multi-usuario, puesto que los distintos usuarios deben conocer los cambios realizados por otros usuarios que puedan afectar a su trabajo [EGR91]. Según Borges y Pino, los mecanismos de consciencia ejercen un papel crucial en las interacciones dentro del grupo [BP99a]. Proporcionan un conocimiento de lo que está sucediendo alrededor de cada miembro.

Al igual que se han distinguido dos tipos de *awareness*, asimismo en esta tesis se caracterizan también los correspondientes *mecanismos de consciencia de grupo* específicos para cada uno de ellos: los *mecanismos locales de consciencia de grupo* y los *mecanismos globales de consciencia de grupo*.

Se define un *mecanismo local de consciencia de grupo* como:

Definición 2.2 (Mecanismo local de consciencia de grupo): Cualquier mecanismo que permita capturar, materializar, mantener y actualizar una representación de la percepción individual que cada miembro tiene acerca de las actividades del grupo (la *consciencia de grupo particular*), con el fin de explotar cualquier interacción entre miembros del grupo

de trabajo, así como fomentar nuevas interacciones que puedan ir en favor de la comunicación y la coordinación entre los miembros, y de ese modo contribuir a la toma de decisiones locales relacionadas con las condiciones de grupo. □

Al fin y al cabo, los aspectos clave de la construcción del conocimiento se sustentan en el proceso de interacción entre individuos, así como en su cognición individual y compartida [Lal01].

Se define un *mecanismo global de consciencia de grupo* como:

Mecanismo global de consciencia de grupo (Collazos et al., 02) *Cualquier mecanismo destinado a capturar, mantener, actualizar e incrementar el conocimiento compartido, así como la distribución del mismo a todos los miembros del grupo, con el fin de contribuir y promover una colaboración real, y por tanto una mayor efectividad en el desarrollo de la actividad grupal.*

Se trata por tanto de algún tipo de soporte (como por ejemplo un agente, como se propone en [CGPO02]) capaz de construir un *conocimiento compartido*, y a partir de él poder inferir propiedades globales, así como establecer consideraciones generales en beneficio del grupo, redundando en un mayor entendimiento de la actividad grupal a los miembros del grupo. En ocasiones puede recurrirse a estrategias de comprensión compartida del problema [CGPO02].

Este es el tipo de aspectos a tener en cuenta y el propósito perseguido con esta componente del contexto de uso. Para determinar qué tipo de connotación (particular o global) se utiliza en cada caso, se hará referencia a los términos y a las definiciones introducidas aquí, facilitando así su distinción.

Y como última componente:

5) La tarea en curso que el usuario está llevando a cabo en un momento determinado mediante el uso del sistema interactivo.

Como se ha podido observar en la descripción de los atributos del contexto de uso, las condiciones dinámicas y cambiantes adquieren una importancia relevante en la caracterización de la situación contextual. En esta tesis se adopta el término **restricción de tiempo real** para hacer referencia a aquellos factores contextuales fluctuantes que describen la vertiente dinámica de cada uno de los componentes del contexto de uso, tratando de caracterizar ese proceso de cambio continuo. Esta distinción es extensible a los cuatro primeros componentes del contexto de uso, tal y como es caracterizado en esta tesis: usuario, equipo, entorno y grupo. En lo sucesivo se utilizará este término para referirse

de forma general a todos los aspectos dinámicos que un sistema interactivo particular se compromete a tener en cuenta en el proceso de adaptación.

Del mismo modo, se adopta la expresión **variación dinámica en el contexto de uso** para hacer referencia a

Definición 2.3 (Variación dinámica en el contexto de uso): Un cambio en el contexto de uso producido por la variación en alguna de las *restricciones de tiempo real*. Se trata por tanto de cambios contextuales de naturaleza dinámica, intrínsecos a la movilidad, las restricciones de los dispositivos móviles y otras *restricciones de tiempo real* preestablecidas en la fase de diseño, que tienen su origen en la evolución natural de la interacción con el sistema, el entorno y el usuario, más allá de estar provocados por el usuario. En la *Hipótesis 2* (véase *sección 1.3*), se plantea que resolver este tipo de variaciones en tiempo de ejecución por el propio equipo reporta numerosos beneficios. □

Este tipo de variación se distinguirá de lo que se define en esta tesis como un *cambio contextual*. Así, con el término **cambio contextual** se hace referencia a

Definición 2.4 (Cambio contextual): Un cambio en el contexto de uso previamente establecido en la fase de diseño, generalmente provocado por el usuario, cuya respuesta o bien conlleva un impacto relevante sobre la IU, fruto de un rediseño, o cuanto menos un replanteamiento de la estructura, disposición y/o contenido de la IU, o bien da lugar a un nuevo entendimiento de la situación grupal, y cuyo tratamiento no puede ser asumido en tiempo de ejecución por el propio equipo de computación. □

Como ya se ha mencionado anteriormente con un ejemplo, en algunos casos la distinción entre *variación dinámica del contexto de uso* y *cambio contextual* estará supeditada a la capacidad de cómputo del equipo utilizado.

2.1.4. Variantes al término de plasticidad

Existen diversos términos para calificar las IUs, todos ellos relacionados con el término de *plasticidad*, aunque con connotaciones diferentes o propósitos menos exigentes que, por supuesto, conviene clarificar y diferenciar a fín de evitar posibles confusiones.

Todos los conceptos presentados en esta sección han sido extraídos del proyecto CAMELEON [CAM04].

En primer lugar, encontramos el término inglés *multi-target*. Hay que tener en cuenta que el término inglés "*target*" –objetivo-, en este caso corresponde a un *contexto de uso*. Es por ello que el término *multi-target* se traduce como *multicontextual*.

Se define una *IU multi-contextual* como:

IU multi-contextual (CAMELEON, 04) *Una IU capaz de soportar múltiples contextos de uso, es decir, múltiples usuarios, plataformas y entornos.*

La diferencia, por tanto, entre *multi-targeting* –capacidad de soportar múltiples targets- y *plasticidad* radica en que mientras el primero se centra en los aspectos técnicos de la adaptación, sin que ello incluya ningún tipo de requisito de usabilidad, la *plasticidad* se ocupa, además, de buscar el mecanismo necesario para controlar y catalogar la usabilidad del sistema mientras la adaptación se está llevando a cabo, a fin de preservarla. En este contexto, el término usabilidad se refiere al conjunto de propiedades expresamente seleccionadas, relacionadas con la facilidad de uso del sistema, las cuales deben ser detectadas y seleccionadas en las fases tempranas del proceso de desarrollo, tal y como se ha expuesto anteriormente (véase *sección 2.1.2*).

Otro término es el de *IU multi-plataforma*, definida como:

IU multi-plataforma (CAMELEON, 04) *Una IU multi-contextual sensible a variaciones de plataforma. Se trata de una IU adaptable y/o adaptativa a múltiples tipos de plataformas. En este caso el entorno y los perfiles de usuario pueden estar o bien modelados como arquetipos, o bien representados implícitamente en el sistema.*

Este término suele referirse también con los términos ingleses *cross-platform*, *multi-device* o también *device-independent user interface*.

Se puede considerar este concepto como una evolución natural del antiguo y también menos restrictivo concepto de portabilidad[38]. Por supuesto, la portabilidad resulta necesaria para alcanzar la plasticidad, aunque no suficiente. De hecho, la portabilidad determina el grado de independencia del código con respecto a la plataforma software, refiriéndose a diferencias en el S.O. y entorno de ejecución entre distintos equipos computacionales pertenecientes a una misma familia (generalmente los PCs o computadores de sobremesa, como por ejemplo entre plataformas del tipo x86, IA64, amd64, etc). Sin embargo, para ofrecer plasticidad no es suficiente con proporcionar abstracción a nivel de código. Se persigue un nivel superior de abstracción, con el fin de superar, además, las diferencias entre aquellas otras capas de software que se sitúan por encima del S.O.,

[38]Capacidad de un sistema de ejecutarse en distintas plataformas, cubriendo los cambios en los recursos hardware y software.

y que configuran el entorno de ejecución de distintas familias de equipos, las cuales son específicas para cada caso. Por ejemplo, en el caso de J2ME para dispositivos compactos se cuenta con la capa de la JVM, la capa de configuración, perfil, paquetes opcionales y finalmente una última capa para las APIs propietarias –si es el caso-, que se sitúa justo por debajo de la capa de aplicación.

Se plantea, por tanto, una meta mucho más compleja y ambiciosa que implica un proceso de reconfiguración de la IU de considerable envergadura. Este proceso requiere cierta mecanización para ser resuelto satisfactoriamente (sin un coste excesivo). Esta es la idea que subyace al concepto de *plasticidad*.

Siguiendo con la clasificación de las IUs, se define una *IU multi-modal* como

IU multi-modal (CAMELEON, 04) *Una IU capaz de soportar múltiples* ***modalidades de interacción****.*

Entendiendo por *modalidad de interacción* las diferentes maneras en que los usuarios se comunican o interaccionan con el ordenador, como por ejemplo por línea de órdenes, a través de menús y formularios, por manipulación directa, mediante interacción asistida, por voz, etc. [PRS+94].

Una *IU multi-entorno* es

IU multi-entorno (CAMELEON, 04) *Una IU multi-contextual sensible a variaciones de entorno. Se trata de una IU* ***adaptable*** *y/o* ***adaptativa*** *a múltiples tipos de entornos. En este caso los distintos tipos de plataformas y los perfiles de usuario pueden estar o bien modelados como arquetipos, o bien representados implícitamente en el sistema.*

Una *IU multi-usuario* es

IU multi-usuario (CAMELEON, 04) *Una IU multi-contextual sensible a variaciones de usuarios. Se trata de una IU* ***adaptable*** *y/o* ***adaptativa*** *a múltiples estereotipos -clases- de usuarios. En este caso los distintos tipos de plataformas y entornos pueden estar o bien modelados como arquetipos, o bien representados implícitamente en el sistema.*

Finalmente, de acuerdo a la definición de estos términos que proporcionan la base del concepto de ***IU plástica***, en el contexto del proyecto CAMELEON [CAM04] se ofrece esta definición alternativa:

IU Plástica 3 (CAMELEON, 04) *Una IU multi-contextual que preserva la usabilidad a través de los distintos contextos de uso.*

En este caso el concepto de *preservación de la usabilidad* se corresponde al enunciado anteriormente en la *sección 2.1.2*. Por lo tanto, de igual modo se refiere a la capacidad de mantener un conjunto de propiedades de usabilidad específico dentro de un rango predefinido de valores.

Todos los conceptos presentados en esta sección han sido extraídos del proyecto CAMELEON [CAM04].

2.1.5. Otros conceptos relacionados

Con el objetivo de medir el grado de adaptación de una IU a los distintos contextos de uso, Calvary et al. introducen las nociones de *dominio de multi-contextualidad* en contraposición al concepto de *dominio de plasticidad*. El primero se refiere al conjunto de contextos de uso para los que una IU es capaz de acomodarse –cobertura de contextos de uso. Al subconjunto de este dominio para el cual se preserva la usabilidad lo denominan *dominio de plasticidad* [CCT01a]. Este concepto puede incluso mostrarse gráficamente mediante un eje de coordenadas representado por el par (plataforma/entorno), tal y como aparece en la figura 2.1.

Bajo este punto de vista definen el *dominio de plasticidad* de una IU como la superficie formada por todas las tuplas (plataforma, entorno) para las que su IU es capaz de acomodarse. Por lo tanto, el *umbral de plasticidad* lo conforman aquellos contextos de uso situados en la frontera del dominio de *plasticidad*, considerado éste como una superficie de una cierta extensión de mayor o menor discontinuidad.

Adicionalmente, introducen el concepto de *discontinuidad de plasticidad* de la forma siguiente:

> *"se produce una discontinuidad de plasticidad cuando se requiere un cambio de contexto caracterizado por un par (plataforma,entorno) que está más allá de la frontera del dominio de plasticidad"*

Así, en la figura 2.1 podemos observar que el contexto de uso denotado como C1 está cubierto por el *dominio de plasticidad* representado en la curva cerrada, mientras el contexto C2 está fuera de su alcance. Por lo tanto, si la IU conmuta desde el contexto de uso C1 al contexto de uso C2 provocaría una *discontinuidad de plasticidad*, lo que se traduce en la poca conveniencia o la imposibilidad de adaptarse a este nuevo cambio, que podría calificarse como una ruptura técnica.

En sintonía con esta visión se puede entender la *plasticidad* como:

Plasticidad 6 (Calvary et al., 01) *La capacidad de superar cambios contextuales sin causar discontinuidades notables (lo más transparente posible).*

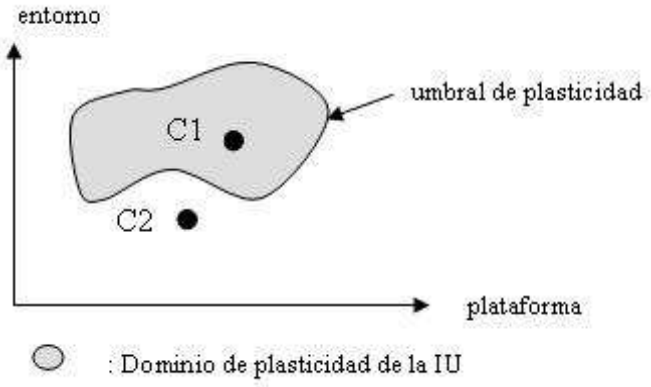


Figura 2.1: Dominio de plasticidad y umbral de plasticidad de una IU.

Relacionado con esta visión de extensión que se le da a estos conceptos, la cual pretende captar la idea de cobertura de la *plasticidad*, sugieren unas métricas para medir la *plasticidad* en relación a su discontinuidad para un sistema interactivo en concreto. Una primera sugerencia consiste en obtener como medida la suma de las superficies cubiertas por el/los distintos conjuntos de contextos de uso, o bien la suma de sus cardinalidades.

En [CCT01b] proponen diversas heurísticas que intentan refinar la medida anterior para obtener una métrica de la *plasticidad*, como son el tamaño de la superficie más grande, el número de conjuntos distintos, las formas o topología de las superficies, etc.

Al margen de todos estos indicadores válidos para razonar sobre la *plasticidad*, consideran también una perspectiva complementaria: la de los usuarios. Con este propósito sugieren dos indicadores: la *frecuencia de contexto* y el *coste de migración entre contextos* [CCT01b].

La *frecuencia de contexto* expresa la frecuencia con la que los usuarios realizan sus tareas en un contexto dado. El *coste de migración* mide el esfuerzo cognitivo y físico que los usuarios tienen que realizar para migrar entre contextos. Afinando un poco más definen el *umbral de coste de migración* como la tolerancia de los usuarios a migrar entre contextos.

En particular, la *migración* entre plataformas constituye una de las posibles manifestaciones de la plasticidad. Se refiere a la transferencia total o parcial de la IU a diferentes recursos de interacción. Puede obtenerse de manera estática (entre sesiones) o dinámica (en una misma sesión). Este concepto puede considerarse como una concreción del concepto de *independencia de dispositivo*, para dos o un conjunto de plataformas concretas.

Al respecto de la migración dinámica entre plataformas, un trabajo interesante es el que se presenta en [MVG06]. Presenta un entorno de realidad virtual donde a través del uso de la manipulación directa, el usuario puede iniciar una migración total o parcial entre diversas plataformas.

Otra vertiente de la plasticidad que surge al trasladar esa facilidad de adaptación al campo de la computación distribuida es el de la *distribución*. Aunque originalmente la plasticidad se concibe como una noción para entornos centralizados, conforme se van abriendo nuevos horizontes**,** este concepto inicial se va extendiendo. Es el caso de la *distribución*. Se propone distribuir los recursos de interacción a través de un cluster[39], incluso tratándose de clusters heterogéneos dinámicos, es decir, conocidos sólo en tiempo de ejecución. Por ejemplo, en el caso de IUs gráficas, el rendering es distribuido si utiliza superficies que son manejadas por diferentes plataformas elementales, abriendo la posibilidad de distribuir la IU entre múltiples dispositivos de interacción. La granularidad de la distribución puede variar desde el nivel de aplicación hasta el nivel de pixel. Igual que en el caso de la *migración*, la *distribución* puede obtenerse de manera estática (entre sesiones) o dinámica (en una misma sesión). Esta idea constituye un primer paso de acercamiento de los beneficios de la plasticidad a la Computación Ubicua, ofreciendo un camino hacia la consecución de esa omnipresencia de la computación que la caracteriza. El primer trabajo que se plantea abordar la *distribución* es el llevado a cabo en el seno del proyecto CAMELEON [CCT+02a], aunque se enfoca desde el punto de vista estático. En [BDB+04] se presenta un marco conceptual que aborda el problema de la plasticidad, migración y distribución dinámicas.

2.1.6. Antecedentes al concepto de plasticidad

Por supuesto, la adaptación es la base de la plasticidad. Este es un concepto mucho más amplio que agrupa diversas propiedades: la adaptación al usuario, la *sensibilidad al contexto* y la *independencia de dispositivo*, aunque en esencia en todas ellas prevalece un mismo principio de adaptación. Son por tanto los objetivos a los que dirigir el potencial de adaptación lo que diferencia dichas propiedades entre sí.

A continuación se presenta una breve revisión de los conceptos relacionados con adaptación a diversos parámetros, conforme han ido surgiendo distintas demandas al respecto.

[39] Conjunto de plataformas elementales. Pueden ser homogéneos o heterogéneos en función de la diversidad de plataformas componentes.

2.1.6.1. Personalización

El término de adaptación, técnicamente *personalización* o también *tailoring* o *customization* hace referencia a la propiedad de ajustar o acomodar diversos aspectos del sistema software de acuerdo a un único *input*: el usuario que está haciendo uso del sistema. La personalización surge de la necesidad de adaptar el contenido y/o el estilo de presentación a las preferencias y necesidades de un usuario, o conjunto de usuarios. Este concepto ha ido evolucionando conforme los nuevos paradigmas de interacción han ido implantándose. Así, el término personalización no se limita a la adaptación al usuario en el acceso a la Web, como podría pensarse. Esta propiedad apareció por primera vez en las aplicaciones interactivas *off-line*, generalmente aplicaciones multimedia.

Entre las técnicas iniciales de personalización o modelado de usuario, existe un gran abanico que abarca desde técnicas muy simples a técnicas mucho más complejas que requieren el uso de reglas y/o implican la generación de contenido en tiempo de ejecución, sin que necesariamente haya habido una interacción explícita con el usuario. Es precisamente esta característica la que diferencia los dos términos utilizados en este campo: *adaptabilidad* y *adaptividad* [DFAB03], que aparecen definidos en la *sección 2.1.3*. En el primer caso, más simple, el usuario es invitado a establecer manualmente sus preferencias respondiendo a una serie de preguntas representativas al inicio de la aplicación. En el segundo caso, es el sistema el que automáticamente debe averiguar cuáles son las preferencias del usuario basándose en la observación de la interacción que éste realiza con el sistema, induciendo su perfil a partir de su patrón de comportamiento. Se trata de un tipo de adaptación mucho más avanzada que en muchas ocasiones utiliza técnicas de la inteligencia artificial –caso en el que se suele hablar de *IUs inteligentes*[40]-, como por ejemplo data mining, u otros algoritmos más complejos. Browne et al. en [BNR90] califican la *adaptatividad* y la *adaptabilidad* como dos propiedades complementarias de un sistema interactivo. Presentan un esquema completo de clasificación, donde el espacio de diseño para la adaptación se describe a través de cuatro ejes ortogonales, basados en estos criterios: objetivo de la adaptación, tiempo, medio y actor que lleva a cabo la adaptación (sistema o usuario). Posteriormente se presenta un esquema de clasificación más completo en [DMKS93], motivado por un análisis detallado de las *IUs inteligentes*.

Sea como sea, una vez obtenidos los aspectos relevantes del usuario, la información generada debe modelarse y representarse a través del llamado modelo de usuario. Es a partir de un modelo de usuario ya representado que se pueden aplicar distintas técnicas

[40]Se considera una IU inteligente como una IU adaptativa que además proporciona otro tipo de servicios inteligentes como son: un sistema tutores (del término inglés tutoring) o un sistema de ayuda inteligente.

para conseguir adaptar no sólo el contenido de la información a presentar al usuario, sino también la IU, e incluso la funcionalidad o comportamiento de la aplicación.

Las técnicas basadas en restricciones están entre las primeras que permiten el desarrollo de interfaces *adaptativas*. Se ha utilizado en sistemas como Amulet [MMM+97]. Para conseguir un grado mayor de *adaptatividad* fue necesario el desarrollo de nuevas técnicas de modelado. En particular, entre las técnicas más empleadas para personalizar el contenido de la información están: *filtrado* (eliminación de información o comportamiento que no es del interés del usuario); *ordenamiento y priorización* (reorganización de la información de acuerdo a las preferencias del usuario); *sugerencia* (realización de sugerencias espontáneas al usuario, presentando información o sugiriendo realizar tareas que se supone que son de su interés).

2.1.6.2. Hipermedia adaptativa

Más adelante, la introducción del World Wide Web ha convertido la hipermedia en el paradigma preferido por el público en general para el acceso a la información. Los sitios Web ofrecen típicamente mucha libertad para navegar a través de un extenso ciberespacio.

Desafortunadamente, esta riqueza en la estructura de las aplicaciones hipermedia ocasiona en muchos casos graves problemas de usabilidad y de comprensión. Los *Sistemas Hipermedia Adaptativos* (en adelante SHA) en general y los sitios Web adaptativos en particular pretenden superar estos problemas proporcionando soporte de navegación y contenido adaptativos. Se está haciendo referencia al campo de la *Hipermedia Adaptativa*, o también *Personalización de la Web*. Se trata de conseguir que los sistemas de información basados en Web se adapten a las necesidades e intereses de usuarios individuales o grupos de usuarios, pretendiendo incrementar su eficiencia en el acceso a la Web. Típicamente, un sitio Web personalizado reconoce a sus usuarios, recoge información acerca de sus preferencias y adapta sus servicios con objeto de satisfacer sus necesidades.

La Taxonomía de Brusilowsky [Bru96] establece una división de las políticas de adaptación en *hipermedia adaptativa* en dos categorías principales: *adaptación en la presentación* y *adaptación en la navegación* y *adaptación de los contenidos*. En concreto, la *adaptación en la navegación* consiste en ayudar al usuario a encontrar en el ciberespacio el camino que necesita, adaptando la forma de asociar los enlaces a las metas y otras características del usuario. Originalmente, esta taxonomía proporcionó un mecanismo para clasificar los distintos SHA. Desde entonces se han desarrollado nuevos sistemas, algunos de los cuales no encajan en la taxonomía existente, y que han forzado a su ampliación, haciéndola más extensa y compleja. En definitiva, esta división causa problemas a un nivel conceptual.

Podemos citar como SHA más representativo, utilizado como referencia en este campo el sistema AHA [BC98], uno de los SHA pioneros, hasta el punto de considerarse como un benchmarck en el dominio.

Otro trabajo muy importante es WUML (Web Unified Modeling Language) [KPRS01] cuyo propósito es definir una metodología que soporte el desarrollo de aplicaciones Web donde también se tiene en cuenta el aspecto de adaptación proporcionando las llamadas reglas de personalización, que en este caso sí llevan a cabo la detección de cambio en el contexto y la consecuente repercusión en el comportamiento de la aplicación. Posteriormente, Brusilowsky propuso un potente mecanismo para el desarrollo de sistemas hipermedia sensibles al contexto [BSW96] que posteriormente utilizó para el desarrollo de sistemas tutores adaptativos.

Existen numerosas áreas de aplicación donde los SHA han ido proliferando. Concretamente, las aplicaciones de comercio electrónico constituyen un ejemplo representativo de sistemas donde la adaptación a nuevos entornos y requisitos constituye un factor crítico. Otra es la de los *Sistemas Hipermedia Adaptativos Educacionales* (SHAE) o *Sistemas Educacionales basados en Web* [Bru99], cuyo objetivo es ofrecer soporte personalizado ajustado a las necesidades y habilidades de cada estudiante/aprendiz en el uso de Internet como herramienta de educación. Una componente esencial en los SHAE es el módulo de evaluación, cuyo propósito consiste en ofrecer una constatación del progreso obtenido en el aprendizaje, con objeto de dirigir su encauzamiento futuro. En general, el proceso de evaluación es dinámico e intervienen tests adaptativos.

Dentro de este campo se ha especializado la rama correspondiente a los *Sistemas Tutores Adaptativos*, que hacen uso de técnicas más complejas que consisten en la implementación de los llamados agentes inteligentes o también agentes adaptativos, cuyo objetivo es adaptar tanto la estructura del curso ofrecido como el contenido a presentar. En este campo, el modelado de tareas de usuario ha permitido en los últimos años añadir con relativa facilidad a las aplicaciones interactivas sistemas que hacen un seguimiento de las acciones del usuario, reaccionando dinámicamente en función de éstas. Estas técnicas han dado paso también al desarrollo de herramientas para la generación automática de sistemas de ayuda y tutores acerca del manejo de interfaces. En esta línea de trabajo se sitúa TANGOW (Task-based learNer Guidance On the Web), un sistema para la creación y seguimiento de cursos adaptativos a través de Internet [CPR99].

Por último cabe mencionar el subcampo correspondiente a los sistemas de recomendación. Estos sistemas tratan de presentar al usuario productos publicados en la Web (películas, música, libros, noticias, páginas Web, etc.), los cuales se presume que son del interés del usuario, o también para la predicción de posibles futuras interacciones. De ma-

nera más general, este enfoque es utilizado para la predicción de respuestas a las cuales otros usuarios previamente han respondido. Así, el perfil de usuario es comparado con respecto a algunas características de referencia, tratándose de una técnica fundamentada en el uso de la experiencia. Actualmente constituye el enfoque más común para la personalización de páginas Web. Se basan en la técnica de filtrado de información, o más concretamente *filtrado colaborativo*.

2.1.6.3. Ampliación del espectro de adaptación

La posibilidad de acceder a Internet desde un número creciente de dispositivos compactos abre nuevas expectativas y campos de exploración. Así, la W3C[41], que entre sus objetivos principales se destaca el del Acceso Universal, desarrolla Composite Capabilities/Preference Profiles (CC/PP), con el propósito de que los usuarios puedan acceder al contenido Web independientemente del dispositivo utilizado. CC/PP proporciona un formato estandarizado para la descripción de la información que permitirá a los dispositivos con acceso Web comunicar de forma efectiva sus capacidades al servidor deseado. Definen los que se conoce como "contexto de entrega", como mecanismo para que tanto las características y capacidades de los dispositivos (Composite Capabilities), como las preferencias de usuario (Preference Profiles) y otro tipo de restricciones establezcan requisitos sobre cómo puede ser mostrado el contenido de forma efectiva en el dispositivo del usuario. Estas especificaciones son las utilizadas por los servidores para responder a las demandas de las aplicaciones.

Posteriormente surge UPS (Universal Profiling Schema) [LL02a], fundamentado en CC/PP. Fue definido con objeto de servir como modelo universal, y proporciona un marco de descripción detallada para distintos contextos. Permite describir otro tipo de elementos involucrados en la adaptación, tales como el perfil del documento, el perfil de conectividad, etc. A través de un proxy entre cliente y servidor tanto el perfil del usuario como el perfil del contenido solicitado son parseados con objeto de resolver el conjunto de restricciones consideradas. SMIL [LO05], por su parte, es una colección de módulos junto con un marco escalable que permite personalizar el perfil del documento a las capacidades del dispositivo. En [LL02a] se presenta una estrategia para la entrega de contenido adaptado en entornos heterogéneos basados en UPS. Tanto UPS como SMIL ofrecen un enfoque eficiente para la producción de contenido adaptado a las restricciones de dispositivos limitados y embebidos.

Cabe destacar también el proyecto de software abierto WURLF[42] (Wireless Universal Resource FiLe), el cual incorpora toda la tecnología necesaria para resolver la problemática

[41] World Wide Web Consortium. http://www.w3c.es/

[42] http://wurfl.sourceforge.net/

del modelado de la información de dispositivos. Además ofrece un framework para la adaptación de aplicaciones móviles. Consiste en un repositorio en formato XML donde se almacenan las capacidades de los dispositivos inalámbricos. De cada dispositivo del mercado, se almacenan datos como su resolución, qué formatos de video soporta, así como numerosos datos técnicos. La información está estructurada en forma de árbol de herencia donde cada nodo representa un dispositivo o una familia de dispositivos, en torno a las capacidades y grupos de capacidades de los dispositivos (sonido, navegación, mensajería, J2ME, pantalla, vídeo, descargas, etc.).

Este proyecto, desarrollado por Luca Passani, empezó en el 2002 y desde entonces se han reunido datos de más de 7.000 dispositivos. Estos datos no siempre los facilitan los fabricantes y en muchas ocasiones son los propios desarrolladores los que averiguan los datos acerca de los dispositivos, que posteriormente añaden al repositorio WURLF. Para favorecer estas colaboraciones se ha desarrollado una plataforma en la que introducir esos datos. Se utiliza un vocabulario XML propio que no está basado en el estándar CC/PP. Surgió con la intención de resolver las carencias de los estándares previos (entre ellos CC/PP y UAProf -User Agent Profile-, también muy utilizado).

Por otro lado, los enfoques comunes para obtener adaptación de contenidos en plataformas móviles heterogéneas son la conversión automática y la especificación explícita del contenido adaptado (conversión manual), las cuales plantean un compromiso entre calidad y esfuerzo de desarrollo y mantenimiento. Estas técnicas se revisan en el *Capítulo 3* (véase *Capítulo 3*; *sección 3.1.4.1*).

2.1.6.4. El salto hacia la plasticidad

La proliferación de dispositivos móviles, así como los últimos avances en la tecnología, entre ellos los relativos a la comunicación inalámbrica, han propiciado el avance en el campo de la *Computación Ubicua* y *Computación Móvil*. En este terreno el concepto de adaptación se hace mucho más ambicioso y entre los *inputs* considerados está no sólo el usuario, sino también el entorno operativo que, en su sentido más amplio comprende cualquier aspecto observable por el sistema software, ya sea a través de sensores o de agentes. La meta es producir aplicaciones interactivas capaces de adaptarse a gran variedad de contextos de uso, de dispositivos de computación y de usuarios, conforme el acceso a la información se hace más universal.

En [Sen07] se proporciona una revisión del estado del arte en estos términos.

2.2. Nueva concepción del problema: *Visión Dicotómica de Plasticidad*

En la *sección 2.2.1* se identifican los retos asociados a la plasticidad, justificando la necesidad de estudiar dos problemas distintos: la *plasticidad explícita* y la *plasticidad implícita*, con dos metas distintas, que es lo se aporta como novedad en esta tesis, enmarcándolo en el enfoque que se ha denominado como **Visión Dicotómica de plasticidad**. En la *sección 2.2.2* se formulan y caracterizan los dos términos, remarcando la diferencia entre ambos, y con respecto a la definición previa de plasticidad, identificando las limitaciones de esta última. En la *sección 2.2.3* se muestran los antecedentes al enfoque basado en esta doble perspectiva. A continuación, se presentan los aspectos tecnológicos y para finalizar las ventajas aportadas por la *Visión Dicotómica de plasticidad*.

2.2.1. Introducción

Tal y como ya se ha evidenciado en el capítulo introductorio de esta tesis, los últimos avances tecnológicos nos facilitan no sólo el acceso a todo tipo de información *"everywhere, everytime"*[43], sino también la realización de cualquier tipo de tarea que pueda ser soportada por un sistema interactivo en entornos cada vez más diversos. Tal y como anticipó Weiser (visionario de la *Computación Ubicua*) [Wei91], los cambios tecnológicos permiten llevar la interacción fuera del "escritorio", para impregnar el ambiente que nos rodea, permitiendo compaginar la computación con la realización de tareas cotidianas, incluso en situaciones de movilidad. Para estar en sintonía con esos entornos, al desarrollador de IUs se le exigen aplicaciones interactivas capaces de adaptarse a gran variedad de contextos de uso y dispositivos de computación sin que por ello se descuide la usabilidad.

En otras palabras, hoy en día los sistemas interactivos deben estar preparados para afrontar una continua y diversa variabilidad inherente a la movilidad, así como la *heterogeneidad de dispositivos*[44], ofreciendo la capacidad de producir sistemáticamente tantas IUs como circunstancias contextuales se establezcan de antemano en una fase temprana del desarrollo. Entre el conjunto de IUs a construir están las que resultan de considerar los diversos equipos de computación en los que se ha decidido implantar el sistema interactivo. Resolver los *cambios contextuales* que implican una remodelación o reconfiguración de la IU a distintas plataformas y familias de dispositivos sin caer en un proceso extremadamente repetitivo plantea un primer reto asociado a un *incremento en la complejidad de diseño de las IUs* (reto planteado ya en el *Capítulo 1* -véase *sección 1.1*-), y por tanto

[43] en cualquier momento y en cualquier lugar [Wei91].

[44] Diversidad y versatilidad en la decisión del dispositivo o plataforma a utilizar, teniendo en cuenta las diferencias significativas en sus características físicas, gráficas y de interacción.

inherente a una fase de diseño. Por otra parte, las capacidades adaptativas deben ser incrementales, es decir, deben evolucionar en tiempo de ejecución conforme las *restricciones de tiempo real* (definidas en la *sección 2.1.3*) varían. Este objetivo consiste en ofrecer una reacción inmediata y dinámica a pequeñas *variaciones* en el contexto de uso, en busca de una mejor armonización con el entorno que rodea la interacción. Este último objetivo plantea un segundo reto asociado a la *acentuación de la necesidad de adaptación dinámica* (planteado ya en el *Capítulo 1* -ver *sección 1.1*). Como ya ha quedado explicitado, este reto atañe a la fase de ejecución del sistema y abarca varios aspectos, desde la detección de los cambios contextuales, hasta la consecuente reacción y adaptación automática de la IU.

En realidad, el propósito de la plasticidad, que podría sintetizarse en: *"velar por la adaptación de la IU a los distintos contextos de uso"* [SL04], no se limita a ofrecer una IU adaptada a las condiciones contextuales y plataforma de computación de partida, sino que involucra un periodo de tiempo y una meta mucho más amplios. De hecho, abarca todo el periodo de utilización del sistema interactivo, dado que la IU puede estar sujeta a una continua variabilidad conforme el sistema atraviesa distintos contextos de uso. En efecto, mientras se está haciendo uso del sistema interactivo puede surgir la necesidad de generar nuevas IUs apropiadas para nuevas circunstancias contextuales (por ejemplo, para adaptarla a una nueva plataforma o dispositivo interactivo). Asimismo, cada una de estas IUs generadas en el transcurso de la interacción deben permanecer activas, puesto que son susceptibles de sufrir nuevas etapas de reconfiguración, o bien pequeños reajustes, intentando satisfacer las diversas necesidades contextuales que puedan ir surgiendo mediante una oportuna adaptación.

Hasta ahora esta doble problemática ha sido acuñada en un solo término: el de *plasticidad de la IU*, introducido por Thevenin y Coutaz en [TC99], como ha sido expuesto anteriormente (véase *sección 2.1.1*). En la presente tesis se identifican, delimitan y acotan dos niveles de operación en el proceso de plasticidad, inherentemente relacionados con las fases de diseño y ejecución. Ambos niveles de operación corresponden a dos sub-propósitos, formulados a través de dos sub-conceptos de plasticidad que constituyen una extensión al concepto de *plasticidad* original mencionado. Son los denominados *plasticidad explícita* y *plasticidad implícita*. Sus respectivos propósitos están relacionados con los dos retos anteriormente planteados, necesarios para que la plasticidad pueda afrontarse adecuadamente y en su totalidad.

Para ser más precisos, el primer reto (*plasticidad explícita*) afronta los *cambios contextuales* -tal y como se introduce en *Definición 2.4* (véase *sección 2.1.3*)- previamente preestablecidos en la fase de diseño. Se trata de cambios en el contexto de uso cuyo

tratamiento comporta cierta envergadura, los cuales no son resolubles en la propia plataforma de interacción. Se requiere por tanto un soporte externo para generar la respuesta adecuada. Algunos ejemplos son: un cambio en el dispositivo utilizado, lo que comporta una reconfiguración de la IU; un cambio en el usuario que utiliza el sistema, el cual posiblemente responde a otro perfil de usuario y por tanto a otras preferencias y necesidades; un cambio relevante en las condiciones del entorno (por ejemplo, pasar de un entorno de interior a un entorno de exterior), que puede comportar tener que activar otros mecanismos de detección del entorno y de su consecuente tratamiento en tiempo de ejecución; la demanda de nuevos contenidos -no disponibles localmente- a lo largo de la ejecución de la tarea en curso (por ejemplo porque el usuario ha cambiado su posicionamiento y la información relacionada con la nueva ubicación se encuentra localizada en un servidor de contenidos); un cambio crucial en la dinámica o en las restricciones de grupo en entornos colaborativos, el cual debe ser compartido por todos los miembros integrantes del grupo (*mecanismo global de consciencia de grupo*). En síntesis, podemos decir que la *plasticidad explícita* es responsable de proporcionar *adaptaciones estáticas* y *complejas* (véase *sección 2.1.2*), ante *cambios contextuales* de diversa índole, como queda plasmado en los ejemplos anteriores. Por el hecho de que se requiere recurrir a un soporte externo, haciendo uso de una petición expresa se le denomina plasticidad "explícita".

El segundo reto (*plasticidad implícita*) afronta las *variaciones dinámicas en el contexto de uso* -tal y como se introduce en *Definición 2.3* (véase *sección 2.1.3*)- preestablecidas también en la fase de diseño. La diferencia es que en este caso, de acuerdo a la *Hipótesis 2* (véase *sección 1.3*) es recomendable resolverlos en tiempo real por el propio equipo de computación, sin requerir el soporte de un sistema externo. En definitiva, se trata de una adaptación *dinámica* y *automática* ante variaciones de diversa índole, que por tanto no requieren la intervención expresa del usuario ni de un soporte externo. Este es el motivo por el que se le denomina plasticidad "implícita".

El tratamiento de este tipo de variaciones implica llevar a cabo modificaciones concretas o pequeños ajustes en la IU de poca envergadura, como por ejemplo un ajuste puntual en alguno de los atributos de la IU (por ejemplo en la luz de fondo del dispositivo móvil; en el orden o en la forma de presentar ciertos elementos de la IU), persiguiendo una mayor satisfacción para el usuario, en ocasiones una *personalización*. El efecto de esos cambios puede incluso pasar desapercibido para el usuario, que simplemente se sentirá cómodo con el uso del sistema a lo largo de toda la interacción. Algunos ejemplos de este tipo de reacciones automáticas son: cambiar la información a mostrar ante una variación en el posicionamiento del usuario; ajustar la luz de fondo a la luz ambiental; ajustar el volumen de una IU por voz al nivel de sonoridad ambiental; alertar al usuario mediante una emisión sonora cuando su pulso corporal alcanza un determinado nivel –para el caso de actividades

deportivas-; aumentar la dificultad de un videojuego cuando el usuario supera una fase; alertar al usuario en situaciones de batería baja y reducir la conectividad inalámbrica -por ejemplo, apagando el bluetooth en un teléfono móvil, para consumir menos batería-; actualizar el *Consciencia de grupo particular* -de acuerdo a la definición *Consciencia de grupo particular*; *sección 2.1.3*- ante un cambio en la percepción individual que un miembro tiene acerca del grupo, como resultado de una interacción efectuada con otro miembro (*Mecanismo local de consciencia de grupo*; véase *sección 2.1.3*), etc.

En conclusión, podemos decir que la *plasticidad implícita* es responsable de proporcionar adaptaciones proactivas, conforme el usuario atraviesa nuevos contextos de uso.

En esta tesis se utiliza el término "dicotomía" [45] para referirse a la separación y delimitación del problema de la plasticidad en estos dos sub-conceptos con dos propósitos distintos. En consecuencia, se denomina ***Visión Dicotómica de plasticidad*** a este nuevo enfoque consistente en separar y delimitar esta doble problemática asociada a la plasticidad, así como el tratamiento a aplicar a cada una de sus metas. Resulta obvio que cada problema identificado requiere unos marcos arquitecturales, unas estrategias y unas técnicas de modelado e implementación distintos, que de forma natural conducen a un tratamiento y a un estudio por separado.

2.2.2. Extensión a la noción de plasticidad: definición de los sub-conceptos propuestos

Como consecuencia de la doble perspectiva presentada en la sección anterior, el enfoque de plasticidad que se propone en esta tesis identifica, acota y delimita los dos objetivos introducidos, que son materializados y formulados como dos sub-conceptos del término de plasticidad. Reciben el nombre de *plasticidad explícita* y *plasticidad implícita*, asociados a los dos niveles de operación identificados: diseño y ejecución, respectivamente. Esos dos sub-conceptos constituyen una extensión al concepto de plasticidad de Thevenin y Coutaz [TC99].

Se define la *plasticidad explícita* como:

Definición 2.5 (Plasticidad Explícita): La capacidad de adaptar (semi)-automáticamente una cierta *IU genérica* ante los *cambios contextuales* preestablecidos en la fase de diseño que requieren la generación o reconfiguración de una nueva *IU específica*, sin descuidar la usabilidad, proceso que se desencadena tras la recepción de una petición explícita procedente de la plataforma cliente en la que se está llevando a cabo la interacción. □

[45] División en dos partes. Método de clasificación en que las divisiones y subdivisiones sólo tienen dos partes.

El concepto de *IU genérica* al que se hace referencia aquí se refiere a una especificación abstracta de la IU, tal y como aparece descrito en IU genérica (véase definición de *IU genérica* en la *sección 2.1.1*).

El concepto de *IU específica* al que se hace referencia en este documento se define como:

Definición 2.6 (IU específica): IU particular y específicamente diseñada para una determinada situación contextual planteada por el usuario, y que es solicitada expresamente ante un *cambio contextual Definición 2.4* durante el proceso de interacción, con el propósito de que se adecue lo más apropiadamente posible a ese nuevo contexto de uso. □

El concepto de *cambio contextual* al que se hace referencia en estas dos últimas definiciones corresponde con el descrito en *definición 2.4* (véase *sección 2.1.3*).

El objetivo a conseguir es el de especificar una descripción abstracta de la IU una sola vez, y a partir de ella derivar las distintas *IUs específicas*, sin descuidar la usabilidad, y tratando con ello de minimizar el coste de desarrollo y mantenimiento mediante una generación (semi)-automática. En concreto, se hace alusión a una generación (semi)-automática, y no simplemente automática, puesto que es ineludible la intervención del desarrollador de IUs para la toma de ciertas decisiones relativas al proceso de generación. No se trata de delegar todo el proceso a una herramienta automática, sino más bien de combinar una explotación automatizada a partir de las especificaciones con un proceso manual por parte de un diseñador humano experto, siguiendo una iniciativa mixta que garantiza la implicación del usuario (diseño centrado en el usuario).

Todos estos conceptos (*plasticidad explícita*, *IU genérica*, *IU específica*) van en concordancia con los lemas con los que se acuñó el término, introducidos también en la *sección 2.1.1.*

En definitiva, la *plasticidad explícita* se asimila a una *adaptación estática* -asociada a la fase de diseño- y *compleja* -implica una reconfiguración o sustitución de la IU en curso por otra específica para la nueva situación contextual-, cuya reacción puede haber sido desencadenada de manera tanto automática (sistema) como manual (usuario) (véase *sección 2.1.2*), ante *cambios contextuales* de diversa índole, generalmente provocados por el usuario. Sin embargo, lo que verdaderamente caracteriza a la plasticidad explícita es el hecho de que no puede ser resuelta de forma autónoma en la propia plataforma de interacción, sino que se requiere un soporte externo, generalmente remoto. En consecuencia, dependiendo de la envergadura de la adaptación, la respuesta obtenida podrá requerir o no la interrupción de la ejecución. En concreto, cuando esas adaptaciones tienen como

objetivo facilitar la conmutación entre distintas plataformas de computación o dispositivos de interacción sin perder el contexto de la ejecución, proporcionan lo que se conoce como *independencia de dispositivo* (tal y como aparece definida en la *sección 2.1.3*). Esta propiedad puede considerarse, por tanto, como una de las posibles manifestaciones de la *plasticidad explícita*.

Esta descripción del sub-concepto de *plasticidad explícita* guarda una marcada analogía con la definición de *plasticidad* aportada en [CCT+02a] en el seno del proyecto CAMELEON, introducido en *Plasticidad 4* (véase *sección 2.1.1*). Esta observación es un indicativo de que, a pesar de los avances obtenidos en el campo de la plasticidad, y a pesar de reconocerse que *"la plasticidad puede ser dinámica y/o estática, así como que puede obtenerse de manera automática y/o manual"* [TC99], hasta ahora no se había ofrecido ninguna definición que reflejara esa doble naturaleza de la plasticidad. Por consiguiente, ambas metas han sido siempre ambiguamente agrupadas en un solo término de plasticidad. Las distintas definiciones utilizadas hasta la fecha han sido siempre más próximas al concepto y propósito de la *plasticidad explícita* propuesta en esta tesis que al de la *plasticidad implícita*, asumiendo que todos los cambios y variaciones en el contexto de uso recibían un mismo tratamiento y un mismo proceso. Si se observa la definición de *plasticidad* que aparece en [CCT+02a]:

> *"capacidad de adaptar una misma* ***IU genérica*** *a múltiples contextos de uso, soportando variaciones en diversas circunstancias contextuales sin descuidar la usabilidad, y al mismo tiempo minimizando los costes de desarrollo y mantenimiento",*

se está refiriendo en todo momento a la fase de diseño, puesto que se habla únicamente de la adaptación que sufre la *IU genérica*, la cual sólo actúa y se maneja en tiempo de diseño. Por otro lado, sólo se hace alusión a los "múltiples contextos de uso", sin analizar ni distinguir la naturaleza de los cambios, ni las causas que provocan la transición entre los mismos. Finalmente, el análisis y control de la usabilidad sólo concierne a la fase de diseño, que es en la que las IUs se generan o remodelan para ofrecer una adaptación *estática* generalmente soportada por un servidor o soporte externo. Los posibles ajustes en la IU que puedan llevarse a cabo en tiempo de ejecución a través de adaptaciones *dinámicas* y *automáticas*, en general no tienen por qué afectar a la usabilidad de la IU en curso, dado que no implican un cambio relevante o efecto suficientemente impactante en la IU como para alterar su nivel de usabilidad. En contrapartida, más bien contribuye a incrementarlo. Esas pequeñas modificaciones favorecen que el usuario se sienta más cómodo con la IU conforme se atraviesan distintas condiciones contextuales, puesto que son ajustes realizados en favor de una mejor armonización con el entorno y el contexto de la actividad en curso, en ocasiones persiguiendo una *personalización*. Se trata de incrementar el *nivel*

de confortabilidad con el sistema (visión presentada en la *sección 1.1*). En cualquier caso, estos cambios repercutirán en favor de la satisfacción del usuario, y de manera indirecta en la usabilidad.

La distinción que se realiza en esta tesis no se reduce a una diferenciación entre las adaptaciones *dinámicas* y las adaptaciones *estáticas*, como ya hicieron Thevenin y Coutaz en [TC99], sino que se plantean dos niveles de operación y dos procesos de naturaleza distinta para llevar a cabo cada una de ellas. En efecto, los ajustes sobre la IU a aplicar ante *variaciones dinámicas del contexto de uso* no puden tener un efecto proactivo si para resolverse se requiriere una remodelación de la *IU genérica* de manera remota. Ciertas modificaciones sobre la IU pueden resolverse de manera autónoma. Este es el motivo por el que se precisa definir un nuevo sub-concepto de plasticidad: la *plasticidad implícita*.

Se define la *plasticidad implícita* como:

Definición 2.7 (Plasticidad Implícita): La capacidad de adaptación incremental de una cierta *IU específica* (en ocasiones un aspecto concreto de la funcionalidad subyacente) a diversas *variaciones dinámicas en el contexto de uso* preestablecidas en la fase de diseño, ofreciendo una reacción automática y su consecuente respuesta en tiempo real, sin que sea requerida la intervención del usuario en ningún momento, mostrando por tanto un comportamiento proactivo. □

Con la definición de *plasticidad implícita* se introduce ya esta distinción, referiendose a otra naturaleza de cambio en el contexto de uso que requiere un tratamiento distinto, para proporcionar una respuesta también distinta. En primer lugar, se expresa que quien sufre la adaptación es la propia *IU específica* que estuviera activa en el momento de producirse una *variación dinámica en el contexto de uso*. Este aspecto es novedoso. También las *IUs específicas* (previamente generadas a partir de una *IU genérica* en fase de diseño) soportan cambios, no tan sólo las *IUs genéricas*. El concepto de *variación dinámica en el contexto de uso* al que se hace referencia corresponde con el descrito en *Definición 2.3* (véase *sección 2.1.3*).

La definición de *plasticidad implícita* constituye una materialización de la siguiente afirmación: *"Las IUs específicas deben soportar variaciones dinámicas en el contexto de uso sin que ello requiera su sustitución o reconfiguración, ni tampoco la intervención de un soporte externo, con el fin de ofrecer plasticidad de manera adecuada y en su totalidad"*, sentando con ello las primeras bases hacia la verificación de la *Hipótesis 2* (véase *Capítulo 1*; *sección 1.3*).

Precisamente, la misión de las herramientas de *plasticidad implícita* es la de dotar a una *IU específica* de la capacidad para soportar este tipo de *variaciones dinámicas en*

el contexto de uso. Esta propiedad incluye un mecanismo para detectar las variaciones, así como un medio de reacción apropiado para llevar a cabo la correspondiente adaptación, como se explica en la *sección 2.2.4*.

Otro punto de distinción a remarcar es el hecho de que ya se hace alusión a una adaptación en tiempo real y *automática*; es decir, una adaptación proactiva. Por último, se hace mención a que los cambios contextuales a tratar a través de la *plasticidad implícita* responden a ligeras variaciones en el contexto de uso, tal y como quedan definidos en *Definición 2.3* más que a *cambios contextuales* de cierta envergadura.

El término "incremental" hace referencia a que es recomendable que las capacidades de adaptación ante variaciones dinámicas no sigan estrictamente un mismo patrón a lo largo de toda la interacción, sino que es deseable que esas adaptaciones evolucionen conforme las *restricciones de tiempo real* varían.

En definitiva, la meta a conseguir es la de ofrecer *proactividad*, esto es, proporcionar la sensación de que el sistema se anticipa a esos pequeños cambios producidos a lo largo de la interacción, por supuesto sin interceptar en la operativa del sistema, ni tampoco en el usuario, siempre y cuando sean abordables por el propio equipo en tiempo de ejecución.

En definitiva, la *plasticidad implícita* se asimila a una *adaptación dinámica* -asociada a la fase de ejecución- y *automática* -sin la intervención del usuario- tanto en la reacción desencadenada como en la respuesta obtenida, equiparándose por tanto a un sistema adaptativo. Sin embargo, y en contraposición con la plasticidad explícita, lo que verdaderamente caracteriza a la plasticidad implícita es el hecho de que se resuelve de forma autónoma, esto es, en la propia plataforma de interacción. En general, el cambio a aplicar comporta una adaptación *simple* (pequeño reajuste de la IU). No obstante, en ocasiones podría implicar una adaptación *compleja* (remodelación de la estructura, disposición o contenido de la IU), siempre y cuando el equipo de computación sea capaz de soportar ese tipo de operaciones en tiempo de ejecución.

En concreto, cuando esos ajustes tienen su origen en una variación en los atributos del entorno, se equipara a lo que se conoce como *sensibilidad al contexto*. Si tienen su origen en una variación en las necesidades, patrones de actuación o preferencias cambiantes del usuario, se equipara a lo que se conoce como *personalización* al usuario. Estas son dos de las posibles manifestaciones de la *plasticidad implícita*.

Ambos tipos de plasticidad son complementarias entre sí. Tal y como ya se ha mencionado anteriormente, para poder afrontar la doble problemática identificada de manera adecuada y en su totalidad, debe proporcionarse un soporte apropiado y específico para cada una. No obstante, son independientes entre sí, esto es, puede coexistir la una sin la

otra, aunque en ese caso el número de parámetros del contexto de uso y de situaciones contextuales factibles de soportar se reduce considerablemente. En cualquier caso, tal y como se presenta en la sección 2.2.4, en esta tesis se propone un modelo de plasticidad (la *visión dicotómica de plasticidad*) y un enfoque tecnológico que integra ambos tipos de plasticidad.

En lo sucesivo, cualquier referencia a los términos *cambios contextual* y *variación dinámica del contexto de uso* harán referencia a las correspondientes definiciones que aparecen en la *sección 2.1.3* (*Definición 2.3* y *Definición 2.4*).

2.2.3. Antecedentes

A pesar de los avances en el campo de la plasticidad, y de que ya en el trabajo que dio origen al término se hace una leve mención a que

"la plasticidad puede ser manual y/o automática ..., estática y/o dinámica" [TC99]

en analogía con anteriores esquemas de clasificación de las *IUs adaptativas* [BNR90], [DMKS93] (véase sección 2.1.6.1.), no se ha enunciado ninguna definición que formalice esta distinción, con el fin de profundizar acerca de las implicaciones respectivas. Tampoco son estudiadas por separado ni vuelve a encontrarse ninguna otra mención al respecto de esta distinción a lo lago de la literatura. La noción de plasticidad *"capacidad de adaptar una misma IU genérica a múltiples contextos de uso..."* [CCT+02a] presenta siempre un enfoque monolítico, sin entrar en ningún caso a discernir qué tipo de adaptaciones abarca o intervienen. ¿Debe entenderse entonces que la plasticidad aplica unas mismas consideraciones, independientemente de si la adaptación involucrada se lleva a cabo de manera local o remota -soportado por un servidor-, de si es manual o automática, de si es estática o dinámica, y en independencia de su envergadura?, ¿o bien todas las adaptaciones deben seguir un mismo patrón operativo desde el momento de la detección del cambio contextual hasta el momento de la ejecución de la adaptación, siguiendo un enfoque rígido e inflexible? ¿Todas esas variaciones de plasticidad se enmarcan bajo un mismo marco de trabajo, unas mismas estrategias y técnicas de modelado? Bajo la opinión particular del autor de esta tesis, existen argumentos a favor de distinguir dos variantes al término de plasticidad -los cuales han sido desarrollados en la sección anterior- a las que aplicar las oportunas consideraciones.

Al enfoque propuesto, basado en una visión biaxial acerca del problema de la plasticidad se le denomina *visión dicotómica de plasticidad*. Se presenta esta distinción formulando los dos sub-conceptos de plasticidad que han sido puestos de manifiesto, rompiendo con

esa visión monolítica. Al mismo tiempo propone una infraestructura de plasticidad estudiando y tratando ambas metas por separado, en niveles de operación diferenciados. En efecto, un soporte de tratamiento común para ambas metas resultaría ineficiente. Este consistiría en efectuar cualquier tipo de adaptación recurriendo a la *IU genérica*, tal y como se anunciaba en [CCT+02a] (véase *sección 2.1.1*). Esta generalización implica que en cualquier caso la adaptación proporcionada por la plasticidad sería siempre *estática* (en fase de diseño) ocasionando que en la mayoría de los casos se interceptara en la operativa del sistema, no resultando transparente para el usuario.

Un trabajo destacable en este sentido es el de de Oreizy et al. [OMT98], que aunque no aborde directamente el problema de la plasticidad, ofrece un estudio y una exploración acerca de los enfoques existentes para construir software auto-adaptativo (*self-adaptive*). Introduce la noción de *open adaptiveness* y *close adaptiveness*, muy cercana a nuestra distinción entre *plasticidad explícita* y *plasticidad implícita*, respectivamente. Así, de acuerdo a esta noción, cuando el sistema incluye todos los mecanismos (detección del contexto, computación y ejecución de la reacción) y datos para llevar a cabo la adaptación por su cuenta se habla de *close adaptiveness*, o lo que es lo mismo, de un sistema *close-adaptive*. En contrapartida, *open adaptiveness* implica que la adaptación se lleva a cabo externamente, ya sea de manera total o parcial. Por lo tanto, podemos asimilar la *plasticidad implícita* con una *close adaptiveness*, y la *plasticidad explícita* con una *open adaptiveness*. No obstante, se trata de una distinción excluyente, es decir, los sistemas son clasificados como *close-adaptive*, o bien *como open-adaptive*. Un sistema no puede ser al mismo tiempo *close-adaptive* y *open-adaptive*. Se asume, por tanto, de nuevo, una visión monolítica. Nuestro enfoque no es excluyente. Se propone complementar ambas naturalezas de plasticidad, potenciando su coexistencia, con el fin de obtener los beneficios de esta doble perspectiva (véase *sección 2.2.5*).

No es hasta 2004, en [BDB+04], que se reconoce la necesidad de un mecanismo externo cuando la adaptación al contexto de uso no puede ser dirigida autónomamente por el propio equipo de computación. Además, por primera vez, se combina un enfoque *close-adaptive* y *open-adaptive* en un mismo marco conceptual de referencia. Así, ante situaciones que no pueden ser manejadas por la propia IU, la cual lleva embebidos mecanismos para auto-adaptación, actúa el mecanismo *open-adaptive*. En este trabajo se propone, también por primera vez, una arquitectura global para soportar la ejecución de IUs plásticas, en concreto para resolver la *migración*[46] entre plataformas en tiempo de ejecución. Se trata de CAMELEON-RT (donde RT representa Run-Time), que incorpora también la manera de interaccionar con esos recursos externos, introduciendo el llamado *open-adaptation*

[46]Transferencia de todo o parte de la IU a diferentes recursos de interacción. Puede obtenerse de manera estática (entre sesiones) o dinámica (en una misma sesión).

manager [Bal04]. Hasta entonces, la plasticidad sólo había sido soportada estáticamente, mediante la computación anticipada de las correspondientes *IUs específicas*. No obstante, la arquitectura presentada tan sólo es concebida a nivel conceptual.

Este es el enfoque más aproximado a nuestra distinción entre una adaptación proactiva y no proactiva considerando, al igual que en nuestro modelo, una combinación de un enfoque *close-adaptive* y *open-adaptive*, de manera complementaria.

Aunque existen algunas herramientas o implementaciones que se aproximan al enfoque que se propone en esta tesis, no obstante, ninguna de ellas aplica una separación entre la adaptación a llevar a cabo en el cliente y la adaptación a llevar a cabo en el servidor. En general, la inteligencia de adaptación reside siempre en el servidor, y este modelo se emplea tan sólo en trabajos orientados a Web. Es el caso del trabajo de Bandelloni en [BP04], donde ante una situación de *migración dinámica*, la plataforma origen envía una petición al servidor de migración, que explota tanto la información de contexto estático como dinámico para llevar a cabo el proceso de mapeo de la presentación. No obstante, tampoco en este caso coexiste un motor de adaptaciones proactivas en la plataforma cliente que complemente la operativa del servidor. Algunos de estos trabajos son revisados en el *Capítulo 3*, y también en la *sección 2.2.4.4*. Otro trabajo, también orientado a Web, que tiene en consideración el perfil del dispositivo cliente a lo largo de la operativa del servidor es el llevado a cabo por Lemlouma y Layaïda del *INRIA Rhône Alpes*[47], que se presentan en el capítulo siguiente. En estos casos tanto el perfil del usuario como el perfil del contenido solicitado son analizados sintácticamente con el objetivo de resolver el conjunto de restricciones consideradas, aplicando un cierto protocolo de comunicación y negociación entre cliente y servidor.

2.2.4. Aspectos tecnológicos

A continuación se proporcionan los aspectos generales del modelo de solución propuesto, sin entrar en detalles de implementación. Se aporta una explicación en profundidad en los Capítulos de explicación del mismo (*Capítulos 4 y 6*).

2.2.4.1. Infraestructura propuesta

Por un lado, el nivel de operación correspondiente a la fase de diseño, denominado *plasticidad explícita*, debe manejar una gran cantidad de decisiones acerca de cómo combinar y entrelazar todas las condiciones y restricciones relativas al contexto involucradas

[47]Instituto Nacional de Investigación en Informática y Automática. http://www.inrialpes.fr/

en esa fase. Este "universo" de parámetros y relaciones debe ser modelado y automáticamente explotado. Por lo tanto, la dificultad relativa a esta fase radica en el manejo de esa multiplicidad de parámetros. Debido a la complejidad inherente, la *plasticidad explícita* requiere ser manejada en un servidor. Este pasará a estar operativo ante una petición procedente de la plataforma cliente. Por otro lado, los reajustes a aplicar en la IU como consecuencia de las *variaciones en el contexto de uso* se llevan a cabo en el propio dispositivo o equipo de computación (el lado cliente de la arquitectura) donde se lleva a cabo la interacción, con el fin de alcanzar la adaptación proactiva que caracteriza la *plasticidad implícita*, conjuntamente con otros beneficios derivados de una mayor autonomía.

Por lo tanto, bajo esta doble perspectiva, una infraestructura de plasticidad que dé soporte a ambas metas debe combinar dos motores distintos, cada uno enfocado hacia un propósito distinto, a enmarcar bajo el modelo de una arquitectura cliente-servidor. Adicionalmente, una arquitectura cliente-servidor proporciona un mecanismo de comunicación entre ambos lados. En este caso se utilizará para resolver la propagación de cambios entre los dos motores, garantizando de este modo la retroalimentación necesaria entre ambos y su consecuente actualización. La meta perseguida es la de mantener ambos motores actualizados durante todo el proceso de interacción sin provocar *discontinuidades de plasticidad*[48].

Conviene puntualizar que teniendo en cuenta que la *plasticidad implícita* se propone resolver las *variaciones dinámicas en el contexto de uso* asumibles en el lado del cliente de manera autonoma, este modelo no tendrá un enfoque centrado en el servidor, y por tanto no será fuertemente dependiente de los recursos de red. En contrapartida, tratará de recurrir al servidor sólo cuando sea imprescindible. Esos casos son: (1) cuando se requiere una remodelación de la IU no asimilable por el equipo cliente debido a un *cambio contextual*; (2) cuando se hace necesario compartir la *Consciencia de grupo particular* debido a un cambio relevante en la percepción individual de las circunstancias de grupo por parte de un miembro integrante, en el caso de entornos colaborativos. Esta dependencia del servidor estará supeditada en parte a la potencia del equipo local para afrontar y soportar los ajustes pertinentes en la IU. En consecuencia, la infraestructura propuesta combina los dos enfoques *open adaptiveness* y *close adaptiveness* complementándolos entre sí.

2.2.4.2. Motor operativo a actuar en el lado del servidor

Para dar soporte a la *plasticidad explícita* se requiere un soporte de desarrollo sistemático de IUs teniendo en cuenta un completo modelado del contexto y de los aspectos

[48] Un cambio contextual que la IU en curso no es capaz de acomodar [CCT01a], es decir, en una misma sesión, sin perder el contexto de la ejecución, evitando una posible interrupción en la operativa del sistema.

estáticos y dinámicos de la IU, así como también las pertinentes restricciones y directrices de usabilidad, en función de las decisiones tomadas al respecto en la fase de diseño. Este motor debe combinar una parte automática con una parte manual, con el fin de que el desarrollador de IUs también pueda intervenir en la toma de decisiones de diseño.

A partir de ahora llamaremos a este motor a actuar en el lado del servidor *Motor de Plasticidad Explícita*. Será el motor responsable de generar (en ocasiones reconfigurar) remotamente una *IU específica* apropiada para una nueva situación contextual que ha sido planteada y expresamente solicitada, al no poder ser asumida localmente por el equipo cliente. En otras palabras, el *Motor de Plasticidad Explícita* resolverá los denominados *cambios contextuales*.

En este caso, tal y como se expone en el *Capítulo 1* (véanse *secciones 1.2* y *1.3*), es ampliamente reconocido que un *enfoque basado en modelos* [Pue97] es el enfoque más adecuado para desarrollar este tipo de soporte al desarrollo de IUs [Lóp05]. Proporciona un mecanismo para diseñar la IU a partir de un conjunto de modelos declarativos que describen no sólo los aspectos estáticos y dinámicos de la IU, sino también otro tipo de factores relevantes, entre los cuales se encuentran los diferentes requisitos de cada contexto de uso, los cuales conjuntamente constituyen el denominado *modelo de interfaz*. Las herramientas construidas siguiendo este enfoque vienen provistas de métodos y útiles que explotan adecuadamente el *modelo de interfaz* para soportar el desarrollo sistemático de la IU. La descripción que estos métodos obtienen de la IU es posteriormente traducida a código directamente ejecutable en una plataforma de computación específica, o bien a algún tipo de lenguaje intermedio (normalmente un lenguaje basado en XML), el cual puede ser interpretado a través de los llamados *visualizadores* (*renderers* en inglés). En resumen, los *métodos basados en modelos* integran el conocimiento de la IU en un método de diseño de la interfaz, proporcionando los formalismos requeridos para la construcción de IUs de manera sistemática.

En conclusión, el *Motor de Plasticidad Explícita* que se propone en la presente tesis consiste en un marco de desarrollo sistemático de IUs plásticas inspirado en el *enfoque basado en modelos*.

2.2.4.3. Motor operativo a actuar en el lado del cliente

Para dar soporte a la *plasticidad implícita* se requiere un motor adaptativo con capacidad dinámica e incremental que permita acomodar fácilmente la IU a las *variaciones dinámicas en el contexto de uso*. A partir de ahora llamaremos a este motor a actuar en el lado del cliente *Motor de Plasticidad Implícita*.

Para ser más precisos, este motor debe proporcionar los mecanismos necesarios para (1) detectar la información acerca del contexto de uso en la que se está interesado; (2) una vez recogida, decidir si debe tener algún efecto sobre la IU, y en qué consistiría; y, en caso afirmativo, (3) aplicar los ajustes necesarios a la IU eficazmente, haciendo reaccionar al sistema en concordancia con el contexto. Este es el proceso a seguir de manera automática y en tiempo real por parte del *Motor de Plasticidad Implícita* para proporcionar una respuesta proactiva ante las *variaciones dinámicas en el contexto de uso*.

En otras palabras, el *Motor de Plasticidad Implícita* es responsable de dotar a una *IU específica* de la capacidad para soportar *variaciones dinámicas en el contexto de uso*. Para conseguir que la IU en curso proporcione esta capacidad de adaptación es necesario embeber de alguna forma en el propio sistema software los mecanismos descritos arriba de detección, recogida, decisión y reacción, a través de una arquitectura software adecuada y flexible. En definitiva, se puede considerar que se trata de hacer "aumentar" al sistema interactivo original con una nueva responsabilidad: la de adaptación al contexto según se haya preestablecido en la fase de diseño.

La tecnología utilizada, la arquitectura propuesta, así como la descripción de cada uno de los mecanismos propuestos para la integración del *Motor de Plasticidad Implícita* en el sistema interactivo se presentan en detalle en el Capítulo 6.

Tal y como se ha enunciado en la *Hipótesis 3* (véase *Capítulo 1*; *sección 1.3*), se presupone que el tratamiento de las *restricciones en tiempo real* constituyen *conceptos transversales*, y que por tanto debe aplicarse una técnica de *Separación de conceptos* para su implementación e integración en el sistema subyacente. Consecuentemente, en la *Hipótesis 4* se asume que la técnica de *Separación de conceptos* más adecuada, y por la que se apuesta es la técnica de *Programación Orientada a Aspectos*. En el Capítulo 7 se aportan los resultados obtenidos de la experimentación llevada a cabo para constatar ambas hipótesis.

2.2.4.4. Mecanismo de retroalimentación

Para que la operación de ambos motores se complemente en todo momento con el fin de obtener los beneficios de combinar los dos niveles de operación, ambos subprocesos deben actuar iterativa y alternativamente, retroalimentándose mutuamente. Se trata de un proceso iterativo no establecido de antemano, sino que depende de las circunstancias contextuales y de la evolución que siga la interacción con el sistema, garantizando una retroalimentación adecuada entre los mapas contextuales (*modelos del contexto*[49]) de ambos lados de la arquitectura. Esta retroalimentación constituye la clave para conseguir que

[49]Especificación de la información relativa al contexto de uso.

el proceso de plasticidad tome en consideración en todo momento los cambios contextuales experimentados durante la interacción, con el fin de obtener la respuesta más apropiada en cada momento. De hecho, la inclusión a ambos lados de la arquitectura cliente-servidor de una adecuada y completa representación del contexto, así como de un mecanismo de propagación de los cambios contextuales constituye un pilar esencial, sobretodo si se tiene en cuenta que es una limitación importante detectada en la literatura sobre el campo. Por este motivo la petición a lanzar por el cliente (quien mantiene una descripción actualizada de la situación contextual) juega un papel crucial.

La emisión de la petición por parte de la plataforma cliente no requiere necesariamente la intervención del usuario. Tanto su generación como su envío pueden estar programados como parte de los componentes del *Motor de Plasticidad Implícita*, llevándose a cabo, por tanto, de manera automática.

Ese proceso se puede resumir de este modo:

> Se asimilan ambos niveles de operación asociados a la **plasticidad explícita** e **implícita** como etapas totalmente **complementarias** de producción -en algunos casos reconfiguración de una IU ya existente- y uso -manejo en tiempo de ejecución- de la IU, los cuales se irán sucediendo iterativa y alternativamente, en función de las necesidades que puedan ir surgiendo a lo largo de todo el proceso de interacción.

Por otro lado, con el propósito de especializar esta infraestructura al ámbito de groupware, con el fin de obtener una colaboración real, es esencial que cada miembro comparta con el resto del grupo su percepción individual acerca del trabajo conjunto. Tal y como se ha introducido en la *sección 2.1.3*, a cada una de estas percepciones individuales se les denomina *consciencia de grupo particular* (según la definición introducida en la *sección 2.1.3*). El servidor es el responsable de reunir y representar una recopilación de las distintas percepciones individuales de cada miembro, hacia la construcción y actualización de una perspectiva global acerca del grupo (el así llamado *conocimiento compartido*), para posteriormente distribuir de algún modo a todos los miembros con el fin de que dispongan de la llamada *consciencia de conocimiento compartido* [CGPO02] , según la definición introducida en la (véase *sección 2.1.3*).

Una vez recopilado el *conocimiento compartido*, éste se incorpora en el proceso de plasticidad. El servidor de plasticidad es también responsable de mantener la consistencia de esa información, con objeto de inferir propiedades globales y consideraciones en beneficio del grupo. El mantenimiento de un *conocimiento compartido* fiable permite conformar una perspectiva global de las restricciones e implicaciones del grupo, con el propósito de que dicha información sea oportunamente explotada durante el proceso de adaptación.

La figura 2.2 muestra la visión de conjunto del proceso de plasticidad que se propone en esta tesis, así como la delimitación entre los dos sub-conceptos de plasticidad. Los componentes que caracterizan el contexto de uso, y que por tanto describen la información contextual a ser enviada al servidor son, de izquierda a derecha: el entorno, el usuario, la plataforma y la tarea en curso.

Figura 2.2: Visión de conjunto del proceso de plasticidad en escenarios individuales.

Como se aprecia en la figura 2.2, el *Motor de Plasticidad Implícita* no es autónomo. Permanece conectado con el servidor por si necesita recurrir a él. De este modo, ambos motores colaboran entre sí en favor de la plasticidad. La simbiosis y retro-alimentación existente entre ellos garantiza una adecuada plasticidad.

Con la incorporación del grupo en entornos colaborativos, la *Consciencia de grupo particular* (*cGp* en la figura 2.3), es también integrada como parte de la información contextual. Por lo tanto, en este caso aparecen 5 componentes. La recopilación de todas estas percepciones individuales conforma el *conocimiento compartido* (*CtoC* en la figura 2.3), que realimenta de nuevo a los miembros integrantes del grupo proporcionando lo que se denomina *consciencia de conocimiento compartido* (*cCtoC* en la figura 2.3).

2.2.5. Ventajas y aportaciones de la visión dicotómica de plasticidad

2.2.5.1. Ventajas generales

- *Balance de carga entre ambos lados de la arquitectura cliente-servidor.* La visión dicotómica fomenta resolver de manera local todos aquellos ajustes o modificaciones en la IU que puedan ser asumidos por el propio equipo, liberando en la medida de

Figura 2.3: Visión de conjunto del proceso de plasticidad en escenarios colaborativos.

lo posible al servidor. Como consecuencia de ello, las responsabilidades derivadas de la adaptación se reparten entre cliente y servidor de manera equilibrada y acorde a las circunstancias, teniendo en cuenta, entre otras cosas, las posibilidades del equipo cliente.

- *Acceso a tantos recursos de red como sean necesarios.* Al tratarse de un enfoque abierto a recursos externos para dar soporte a la adaptación, los mecanismos de adaptación local se combinan con la potencialidad de un servidor, o con tantos recursos de red adicionales como fuera necesario. Las posibilidades de conexión con la que vienen provistos hoy en día los dispositivos móviles, permiten el acceso a cualquier servicio, base de datos o repositorio de información y de modelos.

- *Un mecanismo para la propagación de cambios contextuales.* El mecanismo de comunicación subyacente a cualquier arquitectura cliente-servidor hace posible la propagación de cambios contextuales entre ambos lados de la arquitectura. En consecuencia, ambas componentes -cliente y servidor- se mantienen convenientemente actualizadas, proporcionando retroalimentación en el proceso de plasticidad con respecto al contexto. Este factor contribuye al problema de resolver los cambios contextuales sin discontinuidades, que constituye una limitación importante en la literatura relacionada con el tema, tal y como se ha visto en el *Capítulo 1* (véase *Capítulo 1*; *sección 1.2*).

2.2.5.2. Ventajas de la aproximación presentada desde la perspectiva del servidor

Dado que el enfoque utilizado en el *Motor de Plasticidad Explícita* está basado en el enfoque basado en modelos, podemos citar aquí algunas de las ventajas asociadas a este tipo de herramientas.

- *Abstracción*. La descripción de la IU proporcionada por este tipo de herramientas es de un nivel de abstracción superior al de otro tipo de herramientas de desarrollo de IUs.
- *Sistematización*. Proporcionan métodos sistemáticos de diseño e implementación de IUs donde el diseñador humano también toma parte en las decisiones más relevantes.
- *Completitud*. Proporcionan un soporte completo para todo el ciclo de vida del sistema.
- *Preservación de la usabilidad*. Este enfoque permite fácilmente la integración de una metodología de *diseño centrada en el usuario*, a fin de llevar un mayor control de los parámetros preestablecidos de usabilidad a lo largo del proceso de desarrollo. La intervención del usuario en combinación con la aplicación de directivas de usabilidad apropiadas, también integrables en el marco de desarrollo de IUs *(Motor de Plasticidad Explícita)*, facilita la preservación de la usabilidad.

De hecho, la inclusión o no de este factor es precisamente lo que distingue un marco de desarrollo de IUs plásticas respecto a un marco de desarrollo de IUs meramente multicontextuales.

- *Reutilización*. Es posible la reutilización de modelos y diseños de IU.
- *Minimización del coste de desarrollo*. La producción automática de las distintas *IUs específicas* a partir de una única *IU genérica* evita el sobreesfuerzo de desarrollar y mantener cada una de ellas por separado.

2.2.5.3. Ventajas de la aproximación presentada desde la perspectiva del cliente

La disponibilidad de mecanismos de adaptación dinámica en el propio cliente comporta notables ventajas.

- *Autonomía* para llevar a cabo la adaptación ante variaciones contextuales de naturaleza dinámica, favoreciendo de este modo una reacción proactiva.
- *Acceso inmediato a parámetros locales*, lo que contribuye a la obtención de una *respuesta en tiempo real*.
- *Reducción de la dependencia del servidor y de la bondad de la conexión*. Los fallos potenciales en el sistema de comunicación, en ocasiones inestable, sobretodo si se trata de una conexión inalámbrica, no son determinantes bajo este enfoque; dependiendo de los casos, puede incluso que no afecten en el curso de la adaptación, dado que el equipo cliente podría seguir operando satisfactoriamente satisfactoriamente hasta que se requiriera una nueva reconfiguración de la IU. Este factor contribuye a una mayor robustez en el funcionamiento del sistema.
- *Adaptaciones específicas al dispositivo*. Los mecanismos de adaptación dinámica, que residen en el dispositivo local habrán sido obtenidos y embebidos específicamente para ese tipo de dispositivo. En consecuencia, la adaptación a aplicar se ajustará también a los parámetros específicos y potencialidad del dispositivo en el que se está llevando a cabo la interacción, persiguiendo el máximo rendimiento de la adaptación.

2.2.5.4. Ventajas de su aplicación al campo de *groupware*

- *Autonomía* para llevar a cabo *mecanismos locales de consciencia de grupo*, con el fin de registrar y explotar cualquier interacción entre miembros del grupo de trabajo, así como de fomentar nuevas interacciones que puedan ir en favor de la comunicación y la coordinación entre los miembros.
- *Fomento de la interacción.* La autonomía descrita en el punto anterior, así como la libertad para que los distintos miembros del grupo puedan comunicarse directamente entre sí promueve una mayor interacción y coordinación durante el desarrollo de la actividad grupal. Las condiciones de comunicación no son totalmente dependientes de las condiciones de red, ni tampoco se restringe la capacidad de movilidad. Uno de los objetivos de *los mecanismos locales de consciencia de grupo* es el de materializar y actualizar la representación de la percepción individual que cada miembro tiene acerca de las actividades de grupo (la *Consciencia de grupo particular*; véase *sección 2.1.3*), la cual contribuye a decisiones locales relacionadas con las condiciones de grupo.

- *Un soporte para el conocimiento compartido que contribuye a la colaboración.* La presencia de al menos un servidor, así como el mecanismo de comunicación subyacente a una arquitectura cliente-servidor proporcionan un soporte para reunir las distintas percepciones individuales acerca del grupo (*Consciencia de grupo particular*), y con la información recopilada representar y manejar una visión global del *conocimiento compartido* en el servidor. Esa visión de conjunto permite alcanzar un mejor entendimiento de la actividad grupal (la *Consciencia de conocimiento compartido*), con el propósito de fomentar una colaboración real en escenarios colaborativos distribuidos.

- *Dos niveles de awareness.* Tal y como se ha mencionado en los dos puntos anteriores, se mantiene información de *consciencia de grupo* en ambos lados de la arquitectura: desde una perspectiva individual (cliente) y desde una perspectiva global (servidor). El mantenimiento de ambas perspectivas hace posible su explotación y combinación de la manera más conveniente para cada circunstancia. Estos dos niveles de *awareness* se equiparan a los dos niveles de plasticidad introducidos por la *visión dicotómica de plasticidad* persiguiendo, al igual que en el caso de la adaptación, un balance de carga entre cliente y servidor. En definitiva, este enfoque está en la línea de obtener un equilibrio entre el grado de *awareness* y el uso de la red, propuesto por Correa y Marsic en [CM03].

2.3. Resumen y conclusiones del capítulo

El propósito de este capítulo es el de describir el campo de la *plasticidad de las IUs*, destinado al estudio, descripción y catalogación de la problemática derivada de la diversidad de *contextos de uso*, en la medida que proporciona soluciones económicas y potencialmente ergonómicas, con el propósito de facilitar la adaptación de los sistemas interactivos a esa diversidad. Se trata de un área cuya reciente aparición se justifica en la necesidad de manejar adecuadamente la diversidad de condiciones de uso y circunstancias en las que hoy en día es posible interactuar, definiendo marcos de referencia, útiles o herramientas especializadas para soportar el diseño y desarrollo de IUs plásticas, adaptadas a esta demanda.

En primer lugar, el concepto de plasticidad es ampliamente discutido y analizado. La complejidad del mismo justifica la descripción de los diferentes elementos que componen su concepción. Seguidamente se presentan sus antecedentes, los cuales son introducidos como la evolución natural de los requisitos de adaptación conforme se han ido sucediendo las distintas innovaciones tecnológicas.

En una segunda parte del capítulo se enfatiza la distinción entre los dos retos de plasticidad identificados en esta tesis, justificando la necesidad de ser estudiados por separado y definiendo los dos sub-conceptos de plasticidad propuestos, los cuales constituyen una extensión del término inicial. Al enfoque basado en esta doble problemática se le denominada *Visión Dicotómica de Plasticidad*. Basándose en esta distinción, este enfoque trata de fomentar la repartición de responsabilidades de adaptación entre la plataforma objetivo y un cierto *servidor de plasticidad*, enmarcando la infraestructura de plasticidad en un modelo de arquitectura cliente-servidor (*Hipótesis 1*). La resolución local de cierto tipo de adaptaciones favorece ciertos aspectos como la autonomía, proactividad y reducción de la dependencia del servidor (*Hipótesis 2*).

La incorporación de los aspectos de grupo en la caracterización del contexto de uso; la integración de *mecanismos de awareness* a ambos lados de la arquitectura, a tomar parte en el proceso de plasticidad; y la integración de una visión global del *conocimiento compartido*, a ser manejada y mantenida en el servidor, son los tres pilares para la especialización de esta infraestructura en entornos de *groupware*. Los dos niveles de operación definidos para la adaptación propician el tratamiento y explotación de dos niveles de *awareness*, los cuales son combinados en busca de un equilibrio entre el grado de *awareness* y el uso de la red.

Parte II

Plasticidad Explícita

Capítulo 3

Marcos Conceptuales de desarrollo de IUs Multi-Contextuales Basados en Modelos

"UI design is poised for a radical change in the near future, primarily brought on by the rise of ubiquitous computing, recognition-based user interfaces, 3D, and other technologies, and with it dramatic new needs for tools to build those interfaces."

(Myers et al., 00)

Para diseñar IUs multi-contextuales se requiere un soporte de desarrollo sistemático que construya la IU apropiada a los casos que se planteen, evitando tener que desarrollar manualmente tantas IUs como contextos de uso que en cada sistema se propone cubrir. Esto daría lugar a un proceso extremadamente repetitivo y costoso que, además, repercutiría en gran medida en la usabilidad. Con la explosión de distintos tipos de IUs gráficas, y más tarde con la proliferación de dispositivos, la diversidad a manejar se dispara y eso motiva el desarrollo de nuevas metodologías *genéricas*. La meta a alcanzar es la de flexibilizar el proceso de desarrollo de IUs para afrontar esta diversidad. Actualmente, con la creciente heterogeneidad de plataformas y dispositivos compactos, que dada su versatilidad, amplían aún más la diversidad de situaciones y condiciones de interacción, se impone la necesidad de incrementar aún más el nivel de abstracción en el diseño y desarrollo de IUs, en busca de mecanismos capaces de adecuar automáticamente un mismo diseño a plataformas y entornos distintos.

La idea fundamental de las *metodologías genéricas* consiste en especificar una única IU, genérica y abstracta, suficientemente flexible como para afrontar las múltiples fuentes de variaciones y, a partir de ella, producir tantas IUs como sean necesarias, sin degradar la usabilidad. Se persigue minimizar el coste de desarrollo y mantenimiento. Esta idea, propugnada por autores como Eisenstein, Vanderdonkt y Puerta [EVP01], ha evolucionado hacia dos aproximaciones predominantes: el *enfoque basado en modelo* y el uso de un *lenguaje de IU independiente de dispositivo*, normalmente basado en XML, dada su gran versatilidad. No obstante, existen también otros enfoques interesantes. En la primera sección de este capítulo se analizan las diversas técnicas de especificación y desarrollo automático de IUs, con el propósito de justificar que la tendencia actual hacia un *enfoque basado en modelos* es la más adecuada. Una vez justificado, en la *sección 2* se pasan a presentar con más detalle diversos aspectos del *enfoque basado en modelos*. A continuación, en las dos secciones siguientes se pasa a desarrollar una revisión cronológica de los marcos conceptuales de desarrollo de IUs multi-contextuales, así como diversas herramientas que los soportan, tratando de contrastarlas entre sí con ayuda del marco de referencia unificado. Por último, se hace un rápido recorrido por otras herramientas basadas en modelos, convenientemente clasificadas según la meta propuesta.

3.1. ¿Por qué un enfoque Basado en Modelos?

Antes de pasar a describir en qué consisten las técnicas basadas en modelos se justifica la adecuación de este enfoque frente a otras herramientas de especificación y desarrollo automático.

3.1.1. Requisitos para el desarrollo de IUs Multi-Contextuales

Tal y como se ha ido discutiendo en los capítulos introductorios, el desarrollo de IUs plásticas debe responder a una serie de requisitos, que se enumeran a continuación:

1. *Flexibilidad.* Conviene trabajar a un nivel de abstracción superior al de las herramientas comerciales tipo RAD[1] -o entornos de desarrollo rápido-, de uso predominante en la industria, a fin de separar (1) el conocimiento de la IU, es decir, los aspectos relevantes del sistema interactivo; (2) el diseño de la IU, entendiéndolo como una representación conceptual y genérica de la IU; y (3) la tecnología subyacente y cualquier otro tipo de detalle de implementación y demás requisitos particulares de

[1]Del inglés *Rapid Application Development*. Son Herramientas de diseño para construir directamente la IU.

cada contexto de uso. El objetivo perseguido es el de facilitar que el diseño y la tecnología a utilizar puedan ser alterados independientemente. Los detalles que deben abstraerse abarcan desde las características técnicas (dispositivo utilizado, lenguaje soportado, posibilidades de interacción) hasta los aspectos contextuales, los cuales suelen incluir al usuario y al entorno (véase *sección 2.1.3*).

2. *Reutilización.* Si se consigue aislar el conocimiento que se tiene de la IU de su diseño y de los detalles de implementación subyacentes, efectivamente va a poderse reutilizar no sólo un mismo diseño de IU para distintas plataformas y situaciones, sino también el conocimiento que se tiene de la IU para generar nuevos diseños.

3. *Sistematización.* La meta es ofrecer un soporte que combine un método de diseño sistemático de IUs, bajo una serie de formalismos, con un conjunto de herramientas de generación automática de la IU, donde además se integre el conocimiento de la IU. Una infraestructura de estas características permitiría el desarrollo de IUs de manera sistemática, conforme se presentan nuevas situaciones contextuales, a fin de reducir en la medida de lo posible el coste de producción y mantenimiento de código para las distintas versiones.

Por otro lado, además de esa flexibilidad y mecanización requerida con el objetivo de reducir el sobreesfuerzo de desarrollo, a fin de ofrecer un soporte idóneo para el desarrollo de IUs plásticas, y no meramente multi-contextuales, se debe preservar también la usabilidad. Esto significa que de alguna manera deberá velarse por el cumplimiento de una serie de objetivos de usabilidad previamente establecidos. A día de hoy no se alcanzan los niveles de usabilidad esperados usando herramientas de generación automática. Por lo tanto, añadimos un requisito más en el desarrollo de IUs plásticas:

4. *Implicación del usuario en el desarrollo.* Lo recomendable es seguir una metodología de *diseño centrado en el usuario*[HHN86], garantizando la intervención del usuario con el fin de controlar en todo momento el cumplimiento de los parámetros preestablecidos de usabilidad. Esta supervisión puede ir también soportada mediante la integración de heurísticas, guías de estilo, patrones y demás directivas y principios de usabilidad en el mismo proceso de desarrollo, siguiendo una iniciativa mixta [Hor99] que combina una explotación automatizada con un proceso manual por parte de un diseñador humano experto, satisfaciendo con ello los requisitos de ergonomía intrínsecos a la plasticidad.

3.1.2. Enfoques tradicionales de desarrollo de IUS

Las *herramientas RAD*, o entornos de desarrollo rápido, consisten en herramientas de diseño que facilitan la construcción de la IU final directamente para una plataforma y un contexto de uso determinado, no aplicando por tanto ningún tipo de abstracción. Mediante el uso de estas herramientas, el programador y/o diseñador de la IU, asistido por un IDE (entorno de desarrollo integrado, como puede ser *Eclipse* o *Visual Studio.NET*) va construyendo la IU mediante el paradigma *WYSIWYG*[2], también conocido como *programación visual*. Este proceso consiste en ir eligiendo cada uno de los componentes (*widgets*[3]) que formarán parte de la IU, ajustando sus propiedades y programando el enlace pertinente con la lógica de la aplicación. A pesar de ser unas herramientas muy cómodas puesto que facilitan la programación de la IU, factor que ha permitido que esta fase del desarrollo del software sea mucho más productiva, el producto obtenido proporciona soluciones ad hoc, fuertemente ligadas a la plataforma y a unas necesidades y requisitos funcionales particulares, así como dependientes de un lenguaje de programación o librería de controles determinada. Proporcionan soporte a los aspectos de presentación (parte estática de la IU), pero dejan sin cubrir adecuadamente aspectos del diálogo, los cuales generalmente deben ser codificados a mano. Ejemplos de este tipo de herramientas son los entornos *Visual Basic* (Microsoft Corp.), *Power Builder* (Sybase), *Delphi o C++ Builder* (Inprise Corp.).

Otro tipo de herramientas, como las herramientas de autor (Macromedia Director, Macromedia Flash o Macromedia Dreamwaver) son un claro exponente de editores para la construcción de IUs. Sin embargo, obtener buenas IUs mediante el uso de estas herramientas es complejo. En efecto, la calidad y corrección de la IU así construida queda en manos del "saber hacer" del diseñador que la construye.

Un tercer tipo de herramientas de modelado, las cuales incrementan en cierto grado el nivel de abstracción, como son *Rational Rose* (Rational Corp.), *Together* (Together Soft.), *Argo UML*[4], *System Architect* (Popkin Software), etc., basadas en UML, no proporcionan, sin embargo, mecanismos para la especificación de la IU. El modelado proporcionado por estas herramientas es más cercano al modelado de código que al modelado conceptual, ya que muchas de ellas introducen dependencias del lenguaje destino en la propia especificación, imposibilitando de este modo la portabilidad. Por supuesto, en ningún caso se contempla la posibilidad de incorporar más de un contexto de uso. En este grupo, la

[2]Del inglés *What You See Is What You Get*.

[3]Del inglés *Window Gadget*. Control u objeto gráfico que se dibuja en pantalla y que da soporte a la interacción con el usuario en un entorno gráfico.

[4]Proyecto GNU de código abierto en Java. Disponible en http://argouml.tigris.org

mayoría de herramientas ni siquiera dispone de aspectos específicos para el desarrollo de IUs.

En resumen, aunque este tipo de herramientas pueden resultar válidas para el desarrollo de sistemas mono-contextuales, esto es, sistemas concebidos para ser utilizados exclusivamente en una plataforma y contexto determinados, es obvio que no cubren las necesidades que se plantean en la presente tesis.

3.1.3. Lenguajes de IU independientes de dispositivo basados en XML

Una de las tendencias predominantes en este campo es el uso de lenguajes de IU independientes de dispositivo que, al estar basados en XML ofrecen un alto grado de versatilidad y extensibilidad. A continuación se presentan algunos de los lenguajes de este tipo más utilizados.

UIML (User Interface Markup Language)[5] [APB+99] es un meta-lenguaje que permite describir la IU en términos genéricos. Se trata de un lenguaje libre de suposiciones acerca de la tecnología a utilizar y muy versátil. Al tratarse de un lenguaje basado en XML es posible definir nuevas etiquetas para describir nuevos estilos, creando así un nuevo vocabulario utilizable para generar código específico para diversas plataformas. En síntesis, su técnica consiste en definir una estructura abstracta que aísla las peculiaridades de los distintos dispositivos, a encapsular en diferentes hojas de estilo. En consecuencia, migrar un sistema de una plataforma a otra es relativamente sencillo. Sin embargo, la técnica utilizada no tiene en cuenta consideraciones derivadas del trabajo de investigación en el área del enfoque basado en modelos. Por ejemplo, no existe la noción de tarea, usuario ni modelo.

Las técnicas posteriores construidas para UIML [APSA01] extienden el lenguaje con el uso de transformaciones que permiten al desarrollador la creación de IUs a través de un simple lenguaje que ejecuta en múltiples plataformas. Además, integran ya un modelo de tareas que incrementa el nivel de abstracción de la especificación en UIML con objeto de guiar la IU a distintos dispositivos. Adicionalmente, estos autores han desarrollado un entorno de desarrollo integrado específicamente diseñado para el desarrollo de IUs basado en transformación, el Transformation-based Integrated Development Environment (TIDE) [APSA01].

Una limitación de UIML es el hecho de que al ofrecer exclusivamente un lenguaje simple para definir los distintos tipos de IU, no permite la creación de IUs para distintos

[5] http://www.uiml.org

lenguajes o para distintos dispositivos a partir de una descripción única. Por consiguiente, el diseño de IUs para los distintos dispositivos debe hacerse por separado [SV03].

XForms[6] es un estándar desarrollado por el consorcio W3C[7] como especificación de IUs Web basadas en formularios que aborda la independencia de plataforma. Presenta una descripción de la arquitectura, conceptos, modelo de proceso y terminología subyacente a la generación de formularios Web, basado en la separación conceptual entre el diseño –el propósito- y la presentación concreta de un formulario, remarcando la importancia de esta separación. No obstante, destaca la necesidad de utilizar modelos para soportar este enfoque.

XUL (XML-based User interface Language)[8], fue también desarrollada como una tecnología estándar de la W3C. Se trata de un lenguaje de definición de IUs basado en estándares W3C independientes de la plataforma, y que está asociado con el proyecto Mozilla XPToolkit. XUL no hace distinción entre una aplicación on-line y off-line porque está firmemente enfocada a los dos mundos, consiguiendo que las aplicaciones sean fácilmente portables a todos los sistemas operativos que soportan Mozilla. Proporciona una clara separación entre la lógica de la aplicación, la presentación (hojas de estilo e imágenes denominadas "skins") y las etiquetas de texto específicas del lenguaje. Por supuesto, eso da libertad a alterar la apariencia de las aplicaciones XUL independientemente de la lógica. Las aplicaciones escritas en XUL son muy fáciles de personalizar, y además es compatible con JavaScript, lo que permite la interacción con el usuario.

Una limitación de XUL es que no es aplicable a IUs para dispositivos compactos, sino que está enfocada exclusivamente en IUs gráficas basadas en ventanas. Además, no proporciona abstracciones de funcionalidad de interacción [SV03]

XIML (eXtensible Interface Markup Language)[9] [PE02], cuyo desarrollo inicial se llevó a cabo en los laboratorios de la *RedWhale Software*[10], y está soportado a través de un foro. Se concibió para ser un lenguaje de especificación de IUs universal. La meta de XIML es describir los aspectos abstractos de la IU (e. g. tareas, dominio y usuario) y concretos (e. g. presentación y diálogo) a lo largo del ciclo de desarrollo. El mapeo entre los aspectos abstractos y concretos están soportados [EVP01]. Permite describir completamente la IU, así como representar los atributos y relaciones de los componentes importantes de la misma sin preocuparse acerca de cómo van a ser implementados. De este modo proporciona un mecanismo estándar para que las aplicaciones y herramientas puedan intercambiar datos

[6] http://www.w3.org/MarkUp/Forms/#implementations (1999)
[7] http://www.w3.org/
[8] Mozilla Project. http://www.mozilla.org (2003)
[9] http://www.ximl.org (1998)
[10] http://www.redwhale.com/

de interacción y descripciones de IU, proporcionando la interoperabilidad que lo caracteriza dentro de un proceso de ingeniería de la IU integrado. Hoy en día XIML es probablemente el lenguaje de especificación de IUs más avanzado, que además sirve para especificar otros objetivos como la sensibilidad al contexto. Cumple con los requisitos para ser considerado un lenguaje de especificación de IU universal. Permite especificar cualquier tipo de modelo, elemento de modelo y relación entre modelos. Sin embargo, cabe resaltar que XIML se centra principalmente en los aspectos sintácticos, y no en los aspectos semánticos. Otra limitación es que no existe una herramienta de soporte disponible públicamente.

UsiXML (USer Interface eXtensible Markup Language)[11] [LVM+04], [LV04] es un lenguaje de marcado que describe la IU para múltiples técnicas de interacción, plataformas de computación y modalidades de uso. Se trata de un lenguaje declarativo que captura la esencia de lo que una IU es o debería ser, independientemente de las características técnicas. Describe en un alto nivel de abstracción los elementos constituyentes de la IU de una aplicación: *widgets*, controles, contenedores, modalidades, técnicas de interacción, etc. Al igual que en otros casos, preserva el diseño de las características peculiares de la plataforma física. UsiXML se define mediante un conjunto de XML *schemas*. Cada *schema* corresponde a un modelo distinto.

RIML [Roz97] consiste en un lenguaje basado en XML que combina características de varios lenguajes de marcado existentes (Xforms, SMIL, etc.) en un perfil de lenguaje XHTML. Es capaz de transformar cualquier documento basado en RIML en múltiples lenguajes destino, adecuados para visualizadores vocales o visuales en dispositivos móviles.

Y podrían nombrarse multitud de lenguajes similares, como son Xweb [OJN+00], el enfoque de Schneider y Cordy seguido en AUI (Abstract User Interface) [SC02], AUIML (Abstract User Interface Markup Language), AAIML (Alternate Abstract Interface Markup Lnaguage), XAML (eXtensible Application Markup Lnaguage), AUIL (Abstract User Interface Language), AIAP (Alternate Interface Access Protocol), y muchos más.

A pesar de que han ido surgiendo numerosos lenguajes de IU independientes de dispositivo basados en XML, según Lemlouma et al. en [LL02b], se puede afirmar que ninguno de ellos es totalmente independiente de dispositivo ni que integre todos los aspectos de la adaptabilidad de contenidos. Por otro lado, en general, únicamente están centrados en la independencia de dispositivo, sin entrar en otro tipo de consideraciones contextuales. Además, no existe un soporte sistemático de este tipo de lenguajes, puesto que cada uno utiliza un software específico.

[11] http://www.usixml.org

La variedad existente hace que la adopción de un lenguaje de descripción de IU sea de difícil elección. Según estos autores, esta elección debe realizarse más bien en función de las metas perseguidas, más que en otro tipo de criterios [SV03].

A grandes trazos, XUL es probablemente el menos expresivo [SV03]. En cambio, entre los más expresivos destaca XIML, por el hecho de estar situado en el nivel de metamodelo, adecuado para múltiples contextos. Al tratarse de un lenguaje abierto permite definir internamente un nuevo modelo, elemento o relación. Por otro lado, UIML se destaca entre los más restrictivos. No obstante, es el que está mejor soportado a nivel de software.

3.1.4. Otros trabajos relacionados

Además del enfoque basado en modelos y de los lenguajes de IU independientes de dispositivo, agrupados mediante el calificativo de *metodologías genéricas*, existen otras iniciativas destacables en otras direcciones que tratan de ser más específicas en unos casos, o bien incluso más ambiciosas en otros casos al problema de la especificación y automatización de aplicaciones interactivas no monolíticas (no concebidas exclusivamente para ordenadores convencionales). A continuación se muestran algunos ejemplos.

3.1.4.1. Enfoque dirigido a la producción de contenido adaptado a las restricciones de dispositivo

Los enfoques comunes para obtener adaptación de contenidos en plataformas móviles heterogéneas son la conversión automática y la especificación explícita del contenido adaptado (conversión manual), las cuales plantean un compromiso entre calidad y esfuerzo de desarrollo y mantenimiento. Existen varios proyectos de investigación y aplicaciones comerciales basadas en conversión automática. Un ejemplo es el *Pocket Internet Explorer de Microsoft*[12] que utiliza una tecnología de compresión para reducir páginas. Otras herramientas optan por miniaturizar las páginas Web estándar utilizando una facilidad de zoom. Una técnica muy común es la de paginación, que consiste en trocear la página original en un conjunto de páginas más pequeñas. En ninguno de estos casos se utiliza ningún tipo de conocimiento acerca del dominio del documento, aspecto que mejoraría significativamente la precisión en la conversión. En [BGB03] se propone una arquitectura para resolver online esta conversión de documentos Web basado en la categorización de contenidos. Otras técnicas asumen algún tipo de ajuste del contenido en el lado del servidor, como por ejemplo el proxy de conversión de IBM, que utiliza un fichero de anotación

[12] http://www.microsoft.com/mobile/pocketpc/software/features/internetexplorer.asp

asociado al documento a fin de determinar la relevancia de los distintos fragmentos. Sin embargo, no es asumible que todos los sitios Web proporcionen ese tipo de fichero.

Dentro del mundo de los estándares, cabe destacar el *framework* definido por el World Wide Web Consortium (W3C)[13] denominado "*Composite Capabilities/Preferente Profiles*" (CC/PP)[14], el cual está basado en el *Resource Description Framework* (RDF)[15] [OR99]. Se trata de un *framework* destinado a la especificación tanto de las capacidades de los dispositivos como de las preferencias de los usuarios, especificaciones que utilizarán los servidores especializados para responder a las demandas de las aplicaciones.

Posteriormente surge *Universal Profiling Schema* (UPS)[16], propuesto por Lemlouma y Layaïda del *INRIA Rhône Alpes*[17]. Este marco está basado en las recomendaciones RDF y CC/PP anteriores. Fue definido con el objetivo de servir como modelo universal y proporciona un marco de descripción detallada para distintos contextos. Permite describir otro tipo de elementos involucrados en la adaptación, tales como el perfil del documento, el perfil de conectividad, así como cualquier otra entidad que exista en el entorno del cliente y que juegue un papel en la cadena de adaptación entre el servidor de contenidos y el cliente destino. Precisamente esto es lo diferencia UPS de CC/PP. A través de un proxy entre cliente y servidor tanto el perfil del usuario como el perfil del contenido solicitado son analizados sintácticamente con el objetivo de resolver el conjunto de restricciones consideradas, aplicando un cierto protocolo de comunicación y negociación entre cliente y servidor. La descripción de un contexto en UPS consiste en distintos perfiles almacenados como lenguajes de marcado basados en XML. UPS identifica tres categorías principales de contextos: el cliente, el servidor y la red, para describir también las características de la red. El utilizar distintas categorías a través de distintos tipos de perfiles ofrece una serie de ventajas, las cuales se pueden sintetizar en una optimización de la información a intercambiar. Con objeto de afrontar los cambios de contexto dinámicos presentan dos principios concernientes a una transformación estructural: las transformaciones XSLT y los métodos de *transcoding* (véase *sección 3.5*).

En [LL02a], estos mismos autores presentan una estrategia para la entrega de contenido adaptado en entornos heterogéneos basados en UPS. SMIL[18] [LL03b], por su parte, es un *framework* de adaptación de contenidos para dispositivos embebidos que consiste en una colección de módulos junto con un marco escalable. Permite personalizar el perfil del documento a las capacidades del dispositivo. Tanto UPS como SMIL ofrecen un en-

[13] http://www.w3.org
[14] http://www.w3.org/TR/CCPP-struct-vocab/ (2001)
[15] http://www.w3c.org/TR/1999/REC-rdf-syntax
[16] http://opera.inrialpes.fr/people/Tayeb.Lemlouma/NegotationSchema/index.htm (2002)
[17] Instituto Nacional de Investigación en Informática y Automática. http://www.inrialpes.fr/
[18] http://www.w3.org/TR/smil20/smil20-profile.html (2001)

foque eficiente para la producción de contenido adaptado a las restricciones de dispositivos limitados y embebidos.

En un trabajo posterior [LL04], estos mismos autores enfocan la adaptación automática de contenidos en base a la semántica de los mismos. Para ello definen y se valen de un *modelo independiente del dispositivo*. La arquitectura propuesta, que utiliza el lenguaje *XQuery*[19] (un lenguaje de consulta que permite realizar consultas sobre información expresada en XML) para solicitar información acerca del perfil del dispositivo y entregar los resultados en forma de servicios *SOAP*, se denomina *Negotiation and Adaptation Core* (NAC) [LL03a]. Introducen precisamente la noción de *adaptación de contenido semántico* para referirse a la adaptación de contenido que depende de las preferencias del usuario, del contexto y de la semántica del contenido recibido. Por otro lado, bajo el punto de vista del *modelo de independencia de dispositivo*, un documento consiste en un conjunto de secciones donde cada sección tiene un título y puede ser organizada en diversos niveles de detalle. Además, aplican otras medidas para optimizar la entrega de contenido adaptado, como por ejemplo, solicitando e intercambiando únicamente fragmentos de contenido que tienen una relación directa con la adaptación de contenidos.

3.1.4.2. Extensiones a la notación UML para especificar la IU

UMLi (Unified Modeling Language for Interactive Applications)[20] [PP00] es un proyecto de investigación desarrollado en la Universidad de Manchester que se remonta a 1998. Su principal desarrollador, Pinheiro da Silva propone una ampliación de la notación UML que proporciona conceptos y diagramas para abordar la especificación de la IU, en un intento por atajar las limitaciones que dicha notación presenta para el diseño de IUs. Se trata de una ampliación conservativa, facilitando su utilización por parte de desarrolladores familiarizados con UML. Como derivado de UML, constituye también una notación estandarizada bajo el estilo orientado a objetos. Así, el modelo de dominio en UMLi consiste en un diagrama de clases orientado a objetos clásico. Cabe remarcar que el método está centrado en la especificación de IUs y no en la generación de IUs ejecutables a partir de su especificación. El método utilizado se apoya en la notación formal del lenguaje LOTOS [ISO88].

En cuanto a las incorporaciones que aporta UMLi respecto a UML está la introducción de un nuevo diagrama: el *diagrama de presentación de la IU*, el cual utiliza una serie de primitivas que Pinheiro ofrece para la especificación abstracta de la IU. Estas primitivas, denominadas *Primitive Interaction Objects* son las siguientes: *Freecontainer, Container,*

[19] http://www.w3.org/TR/xquery/
[20] http://www.cs.man.ac.uk/img/umli

Inputter, Displayer, Editor y *ActionInvoker*. Por otra parte, el modelo de tareas en UMLi es especificado utilizando una extensión de los diagramas de actividad, a los que se les introducen una serie de estereotipos para plasmar la interacción de flujo entre los objetos. Éstos son: <<presents>>, <<interacts>>, <<cancels>> y <<activates>>. Por último, se introducen estereotipos también a los estados en los diagramas de actividad. El método propuesto por UMLi es soportado por ARGOi[21], una extensión del entorno ArgoUML[22] de código abierto.

UMLi constituye el primer intento serio de extender UML para capturar la IU de un modo formal. Sin embargo, centra su radio de acción principalmente en las IUs basadas en formularios e IUs WIMP[23]. Para otro tipo de IUs gráficas, las primitivas propuestas pueden no ser las adecuadas. Además, la fina granularidad de los conceptos manejados conlleva limitaciones importantes de escalabilidad. Los problemas de tamaño medio se vuelven complejos y tediosos de especificar, lo cual dificulta en gran medida su adopción en entornos de desarrollo industriales.

Por último, cabe resaltar que la gran aportación de UMLi es la de agrupar y combinar, por un lado el desarrollo automático de IUs mediante un entorno MB-UIDE (véase *sección* siguiente), y por otro el modelado de la aplicación subyacente mediante UML, proporcionando de este modo la integración necesaria entre la IU y la aplicación, atacando con ello una de las limitaciones clave del enfoque basado en modelos (véase *sección 3.2.4*). Además, en los casos donde el entorno MB-UIDE ya soporta el modelado de la IU con UML, la integración es directa. Como desventaja se puede apuntar que la curva de aprendizaje de UMLi no es ni mucho menos trivial.

Otros intentos de extensión de UML son OVID (Object, View and Interactio Design) [RBIM98] desarrollado por IBM, que fue el primer intento de extender UML para la especificación de IUs orientadas a objetos. En este caso la especificación es poco formal y la correspondencia entre las vistas y la implementación se deja totalmente abierta a criterio del diseñador. WISDOM (Whitewater Interactive System Development with Object Models) [NC00] es también una propuesta metodológica para el desarrollo de IUs basada en UML, notación utilizada para la representación de los modelos propuestos. Entre las distintas propuestas se presenta una adaptación del modelo de tareas ConcurTaskTrees a UML mediante la creación de un perfil de UML.

Por último, dentro de este apartado mencionaremos la metodología IDEAS (Interface Development Environment within O**AS**IS) [LGR00]. Esta metodología cubre todo el ciclo

[21] http://www.cs.man.ac.uk/img/umli/download2.html
[22] http://argouml.tigris.org (2003)
[23] del inglés Window, Icon, Menu, Pointing device, que denota un estilo de interacción que utiliza estos elementos. Desarrollado en Xerox PARC en 1973.

de vida del desarrollo de IUs centrado en el usuario. Permite la especificación de la IU tanto a través de una serie de diagramas basados en UML como formalmente usando el lenguaje de especificación OASIS [LRSP98]. IDEAS también plantea una especificación abstracta de la IU que es traducida al lenguaje XUL (Mozilla Project) –anteriormente introducido- para su visualización.

3.1.5. El enfoque Basado en Modelos

El desarrollo de IUs basado en modelos, soportado en lo que comúnmente se conoce como *MB-UIDE*[24], consiste en un mecanismo para diseñar y desarrollar IUs a través de una especificación de la misma a un alto nivel de abstracción utilizando *modelos declarativos* y su posterior explotación hasta la obtención de las distintas IUs requeridas. Dichos modelos describen, representan y formalizan explícitamente no sólo los aspectos estáticos y dinámicos de la IU, sino también otro tipo de facetas, artefactos y factores relevantes involucrados en el desarrollo de una IU, entre los cuales se encuentran los diferentes requisitos de cada contexto de uso. La especificación de toda esta información relativa a la IU a través de modelos declarativos constituye su *representación conceptual*, que implica la creación de una base de conocimiento con la descripción de todas estas componentes de la IU. La unión de todos estos modelos, juntamente con el diseño de la IU a distintos niveles de abstracción, constituye lo que se denomina *modelo de la interfaz*[25].

Las herramientas basadas en modelos se pueden considerar como la unión de dos componentes principales: (1) un entorno interactivo que integra las herramientas de modelado de la IU, así como un espacio de almacenamiento de todo ese conocimiento de la IU (el *modelo de interfaz*) en un método de diseño de la IU; y (2) un conjunto de métodos, útiles, directivas y herramientas subyacentes encargados de explotar los modelos, y que proporcionan los formalismos requeridos para soportar el desarrollo sistemático de IUs.

La descripción que estos métodos obtienen de la IU es posteriormente traducida a código directamente ejecutable en una plataforma de computación específica, o bien a algún tipo de lenguaje intermedio (normalmente un lenguaje basado en XML), el cual puede ser interpretado a través de los llamados *visualizadores* (*renderers* en inglés), que no son más que generadores de código que automatizan la generación total o parcial de la IU de forma sencilla, inclusive cuando los requisitos cambian. En efecto, un mismo diseño declarativo puede ser convertido en código ejecutable para distintas plataformas o lenguajes, sin necesidad de un rediseño total de la IU.

[24]Del inglés *Model-Based User Interface Development Environment*.

[25]Descripción formal, declarativa e independiente de dispositivo de la representación conceptual de la IU, la cual debe ser expresada a través de un lenguaje de modelado.

En resumen, este tipo de herramientas reúnen en mayor o menor grado todos los requisitos definidos anteriormente: flexibilidad y un alto nivel de abstracción al utilizar especificaciones de la IU e integrar la consideración de distintos contextos y circunstancias, sistematización, reutilización de modelos y diseños de la IU, y por último, la posibilidad de intervención e implicación del usuario al combinar procesos manuales (herramientas de modelado) y automáticos (herramientas de explotación de modelos y generación de la IU). Este último aspecto garantiza la preservación por parte del experto humano de los criterios de usabilidad y calidad esperados en la IU final. De hecho, una de las características del *enfoque basado en modelos* es precisamente la facilidad de integración del *diseño centrado en el usuario*.

3.1.6. Justificación

Las herramientas de modelado mencionadas, previas a las herramientas basadas en modelos (Rational Rose, Together, Argo UML, System Architect, etc.), tienen una fuerte penetración en el mercado como herramientas comerciales. Sin embargo, adolecen de soporte para la especificación de IUs, siendo su principal uso la documentación y modelado a nivel de diseño, el cual resulta dependiente del lenguaje, la plataforma y la arquitectura destino. Las propuestas basadas en lenguajes de IU independientes de plataforma consideran modelos orientados a la interacción, aunque están principalmente centrados en el desarrollo de IUs para dispositivos gráficos con distintos tamaños de pantalla, sin proporcionar el soporte necesario para IUs multi-modales.

La proliferación de lenguajes de descripción de la IU resulta en una variedad de dialectos basados en XML, los cuales no son ampliamente utilizados, además de que no permiten interoperabilidad entre las herramientas que han sido desarrolladas en el área [MLV+05].

Las herramientas basadas en modelos son en la actualidad la piedra angular en la investigación para la especificación y desarrollo de IUs basadas en modelos. Además, van en consonancia con la tendencia en el desarrollo de software siguiendo el enfoque *Model-Driven Architecture*, mencionado más adelante.

Capturan explícitamente el conocimiento acerca de la IU en un nivel apropiado de abstracción y a través de un método apropiado se estructura la definición y uso de los modelos subyacentes por etapas. La principal aportación del enfoque basado en modelos es el de flexibilizar el diseño de IUs incrementando la reutilización de diseños y códigos para la IU, obteniendo IUs que son más usables en diferentes contextos de uso.

No obstante, una separación total entre la funcionalidad y la IU tampoco resulta recomendable, si de lo que se trata es de construir aplicaciones completas. Ambas componentes

aparecen en la aplicación final, y por tanto se debe velar por mantenerlas sincronizadas. En esta línea, OVID [RBIM98], UMLi [PP00] y Wisdom [NC00] son trabajos enmarcados en la línea de extender la notación UML (usada para la especificación de aspectos estructurales y de comportamiento) con aspectos para la especificación de IUs. Sin embargo, si se pretende producir código para la aplicación final a partir del modelado, es necesario poder disponer de una semántica precisa en el lenguaje de modelado. A día de hoy, UML no tiene tal semántica estandarizada, lo cual dificulta la creación de traductores y generadores de código. Esta y otras limitaciones permanecen sin resolver en el campo de las técnicas basadas en modelos (véase *sección 3.2.5*). Aún así, constituyen el enfoque actual más idóneo, y la solución a esos inconvenientes se encamina más bien en la línea de mejorar los métodos basados en modelos que en la de proponer un nuevo enfoque distinto.

3.2. Otros aspectos del enfoque Basado en Modelos

Una vez justificada una mayor adecuación de este enfoque frente a otras alternativas para afrontar la generación de IUs multi-contextuales, se presenta su origen, y se detalla su consistencia, método y componentes de que constan este tipo de herramientas con el propósito de llegar a tener una visión crítica del mismo, una vez haber identificado sus ventajas e inconvenientes.

3.2.1. Contextualización

El *enfoque basado en modelos* constituye una de las iniciativas del llamado MDE (Model-Driven Engineering), término utilizado en la comunidad investigadora internacional para hacer referencia al uso sistemático de modelos como artefacto de ingeniería principal a través de todo el ciclo de vida del producto software. Cabe señalar que la iniciativa MDE más conocida en el enfoque de diseño de software es el denominado *Model-Driven Architecture* (MDA) [Bro04], cuyo patrocinador es el grupo Object Management Group[26] (OMG), que mantiene marca registrada sobre MDA. El enfoque MDA, oficialmente lanzado en 2001, se ha concebido para dar soporte a la ingeniería dirigida a modelos para los sistemas software. Uno de los principales objetivos de MDA es separar el diseño de la aplicación, que alberga los requisitos funcionales de la misma utilizando un modelo independiente de la plataforma (el llamado PIM, del inglés Platform-Independent Model), de la arquitectura y tecnologías utilizadas para su construcción, es decir, la infraestructura a través de la cual se hacen efectivos los requisitos no funcionales. Igual que en el

[26]Consorcio originalmente creado para establecer estándares para sistemas orientados a objetos distribuidos, y que está actualmente enfocado en el modelado y en los estándares basados en modelos.

enfoque basado en modelos, MDA proporciona un conjunto de guías para estructurar las especificaciones expresadas a través de modelos. El objetivo perseguido es que el diseño y la arquitectura puedan ser alterados independientemente, asegurando que el PIM sobrevive a cambios que se produzcan en las tecnologías de software utilizadas. La traducción entre el PIM y un determinado modelo específico de plataforma se realiza normalmente utilizando herramientas automatizadas de transformación de modelos.

Podría decirse que el *enfoque basado en modelos* constituye una concreción del enfoque de diseño de software MDA en el mundo de la interacción. En este caso lo que se propone obtener a partir de una descripción basada en modelos no es la propia aplicación, sino su interfaz.

Para que una herramienta de IU pueda ser considerada un entorno de desarrollo de IUs basado en modelos se requiere el cumplimiento de, al menos, estos dos criterios:

- El entorno debe incluir un modelo de alto nivel, abstracto, formal y explícitamente representado –declarativo- acerca del sistema interactivo a desarrollar. Se trata de un modelo de tareas o bien de un modelo del dominio, o ambos [Sch96].
- El entorno debe explotar una relación clara y automatizada entre el modelo anterior y la IU operativa deseada. Esto significa que debe haber algún tipo de transformación automática, ya sea generación basada en conocimiento o una simple compilación para implementar la IU final [Sch96].

Las distintas herramientas basadas en modelos pueden ser comparadas con respecto al número y expresividad de sus modelos, o bien por las herramientas de diseño que se encargan de la explotación de la información. Existen algunos trabajos enfocados en la comparación de distintas propuestas y herramientas. Algunos ejemplos son [SE96a], [Sch96], [GFP+98] y [Pin00].

3.2.2. Un poco de historia

Las técnicas de modelado de la IU ya han sido explotadas en profundidad para GUIs[27] de sobremesa tradicionales en [AH95], [Amb04], o también en [Pue96], por nombrar algunas donde el concepto de objeto de interacción abstracto ha sido aplicado con éxito para este tipo de IUs. Con la proliferación de nuevos dispositivos surge la necesidad de incorporar también el manejo de restricciones de plataforma. Podemos citar SUIT [PCD92] como una de las primeras iniciativas en emplear una definición de IU única que puede ser procesada

[27] Del inglés Graphical User Interface.

en múltiples plataformas. Otro ejemplo posterior es Galaxy [Amb04]. Sin embargo, ninguno de estos dos ejemplos corresponde a una aproximación verdaderamente basada en modelo, puesto que no proporcionan el nivel de abstracción necesario para alcanzar los objetivos perseguidos. Esta falta de abstracción, particularmente en la relación entre estructura de tareas y restricciones de plataforma limita la flexibilidad de estas aplicaciones.

Los UIMS (User Interface Management Systems) [Mär90] -sistemas gestores de IU- constituyeron la primera herramienta basada en modelos que añadió una componente de plataforma al modelo de IU. Este término fue acuñado para sugerir su analogía con los sistemas gestores de bases de datos. Además, introdujo también la capacidad de decisión dinámica. En particular, las propuestas que fueron surgiendo a posteriori satisficieron estas características. Un ejemplo es Cicero [AH95], que consiste en un sistema mediador (árbitro) que intenta asignar recursos dependiendo de restricciones conocidas en tiempo de ejecución. No obstante, su aproximación basada en modelos está exclusivamente enfocada a IUs multimedia. En general, los sistemas UIMS sólo soportan la generación automática de IUs WIMP para ordenadores de sobremesa y portátiles [CCT+02a]. Los UIMS clásicos tienen un nivel de abstracción muy bajo, tal y como ocurre en la programación visual, donde se manipulan directamente los elementos de implementación de la IU. La primera generación de MB-UIDEs, caracterizada por la capacidad de ejecutar IUs representadas de manera declarativa, apareció como mejora a los sistemas UIMSs.

Esta categoría de herramientas enfatizó la generación totalmente automática de la IU final y se centró en modelar el dominio de la aplicación subyacente utilizando un modelo de dominio. Un intento por incrementar ese nivel de abstracción se refleja en la herramienta UIDE (User Interface Design Environment) [FKKM91], una de las herramientas MB-UIDE de primera generación. Esta herramienta utiliza una base de conocimiento para asistir el proceso de diseño, evaluación e implementación de la IU. No obstante, el nivel de abstracción en la descripción de la IU deja mucho que desear. El siguiente paso fue la aparición de la segunda generación de MB-UIDEs, en las que los mecanismos para describir la IU son de un mayor nivel de abstracción y el proceso de diseño de la IU es incremental. Los desarrolladores ya son capaces de especificar, generar y ejecutar IUs. Otro reflejo de un avance destacado es que las herramientas tienen un conjunto más diverso de metas y aspectos a considerar, puesto que se explota un rango mucho más amplio de modelos que permiten especificar aspectos específicos de la IU. Además, se aprecia una mejora notable en la calidad y variedad de las IUs generadas. Los métodos empleados permiten explotar completamente la información recogida durante la fase de análisis de requisitos.

En esta revisión de las herramientas de modelado es ineludible resaltar las técnicas desarrolladas por Vanderdonckt et al. Así, una de sus principales aportaciones al campo de

la especificación de IUs es la introducción de los conceptos *Objetos Abstractos de Interacción*[28] (AIO, del inglés *Abstract Interaction Object*) y *Objetos Concretos de Interacción*[29] (CIO, del inglés *Concrete Interaction Object*) [VB93]], que se han revelado como abstracciones muy útiles a la hora de abordar el problema de la IU. Sin embargo, una limitación de esta técnica es que depende únicamente de los modelos de plataforma, presentación y tareas. Para muchas aplicaciones es esencial modelar también a los usuarios, o bien las características del dominio de las tareas. Por otro lado, se requiere todavía el esfuerzo por parte del diseñador de personalizar la IU genérica a cada tipo de plataforma. El siguiente objetivo es, por lo tanto, conseguir que la estructura de presentación correcta sea generada por el mismo sistema, dado un conjunto de restricciones. En este sentido estos mismos autores proponen en [VFOM01] técnicas que amplían la anterior para dar solución a este aspecto, las cuales se basan en la definición de dos nuevas abstracciones: la *Ventana Lógica*[30] y la *Unidad de Presentación*[31], esta última descomponible en una o más *ventanas lógicas*. Utilizando esta jerarquía construyen herramientas de diseño automatizadas capaces de generar varios modelos de presentación optimizados según la plataforma, a partir de un modelo de presentación inicial genérico. Esta idea ha sido utilizada en herramientas como TRIDENT (Tools foR an Interactive Development ENvironmenT) [BHL$^+$95] y TIMM (The Interface Model Mappped) [PE99], [EVP00]. En concreto, TIMM resuelve la selección automática de elementos de interacción, ofreciendo un conjunto de alternativas. Adicionalmente incluye también una componente adaptativa para aprender las preferencias del diseñador. TIMM es una de las herramientas que forman parte del entorno integrado de desarrollo de IUs MOBI-D (Model-Based Interface Designer)[Pue96]. Es la encargada de crear los mappings entre los diversos modelos formales [EP00]. El desarrollo de estas herramientas supuso un paso importante en el camino hacia la abstracción.

Otro aspecto a considerar tan o más importante es el de la personalización. Cabe decir que en la actualidad no son demasiados los trabajos que consideran el modelado de usuario para soportar el diseño de aplicaciones multi-plataforma. Uno de los primeros trabajos donde sí se incorporó fue Hippie [OS00], un prototipo para asistir a los usuarios cuando acceden a información sobre museos, ya sea a través de Web, o bien in situ con una PDA (Personal Document Assistant). Un trabajo posterior es el de Marucci y Paternò en [MP02], que integran las tecnologías móviles y el modelado de usuario con el modelo de

[28] Una abstracción de un conjunto de CIOs con respecto a un conjunto de propiedades que permiten la descripción de la IU de una forma abstracta, independiente de la plataforma y otras propiedades.

[29] Constituyen una cristalización de los AIOs para una modalidad y condiciones contextuales concretas.

[30] Cualquier agrupación de AIOs, como son una ventana física, un área en una subventana, un cuadro de diálogo o un panel.

[31] Se define como un entorno de presentación completo requerido para llevar a cabo una tarea interactiva particular.

tareas en tiempo de diseño. Por primera vez se utiliza el modelo de tareas mediante una notación ConcurTaskTrees [Pat99] en el desarrollo de aplicaciones multi-plataforma.

Hoy en día existen multitud de aproximaciones basadas en modelos. Sin embargo, en la mayoría de los casos se trata tan sólo de proyectos académicos que han sido poco probados en la construcción de sistemas reales. Una excepción son los proyectos comerciales como OlivaNova o VisualWADE, comentados más adelante (véase *sección 3.5.*).

3.2.3. Descripción del proceso de diseño

La arquitectura general dentro del diseño de IUs basado en modelos aparece reflejada en la figura 3.1. En primer lugar, el desarrollador de la IU formaliza los distintos aspectos de la IU aplicando un proceso de diseño y utilizando una herramienta de modelado integrada en un entorno interactivo. Como resultado se obtienen los distintos modelos declarativos que representan el conocimiento que se tiene de la IU. Las herramientas de modelado suelen ser herramientas visuales donde el usuario hace uso de una notación gráfica que permite la especificación de los distintos aspectos de manera sencilla. El entorno interactivo de modelado crea y modifica los modelos que representan el conocimiento que se tiene de la IU, haciendo uso durante todo el proceso de una base de conocimiento donde se recopila la experiencia adquirida por los desarrolladores, que puede consistir en guías de estilo [SM86], [Shn92], [IBM93], heurísticas y patrones [MLGR02], [MLML03], que guiarán la transformación de los modelos en código. Todas estas orientaciones se pueden agrupar e identificar con el nombre de *directivas de interacción.*

Un motor de generación automática genera el código de la IU mediante la explotación de los modelos creados por el desarrollador de la IU y asistido por la experiencia recopilada sobre el diseño de IUs usables. En resumen, el propósito de estas técnicas es explotar toda la información recopilada a través de modelos declarativos para ayudar a los diseñadores a producir diferentes IUs y a tomar decisiones, de acuerdo a los requisitos de cada contexto de uso.

Como se aprecia en la figura, esta aproximación separa el desarrollo de la parte funcional de la aplicación de la generación de su IU. Ambas partes, una vez desarrolladas, son unidas entre sí para construir la aplicación final. Mientras que la IU es construida por el desarrollador de la IU, el código de la aplicación es creado por el desarrollador de la aplicación.

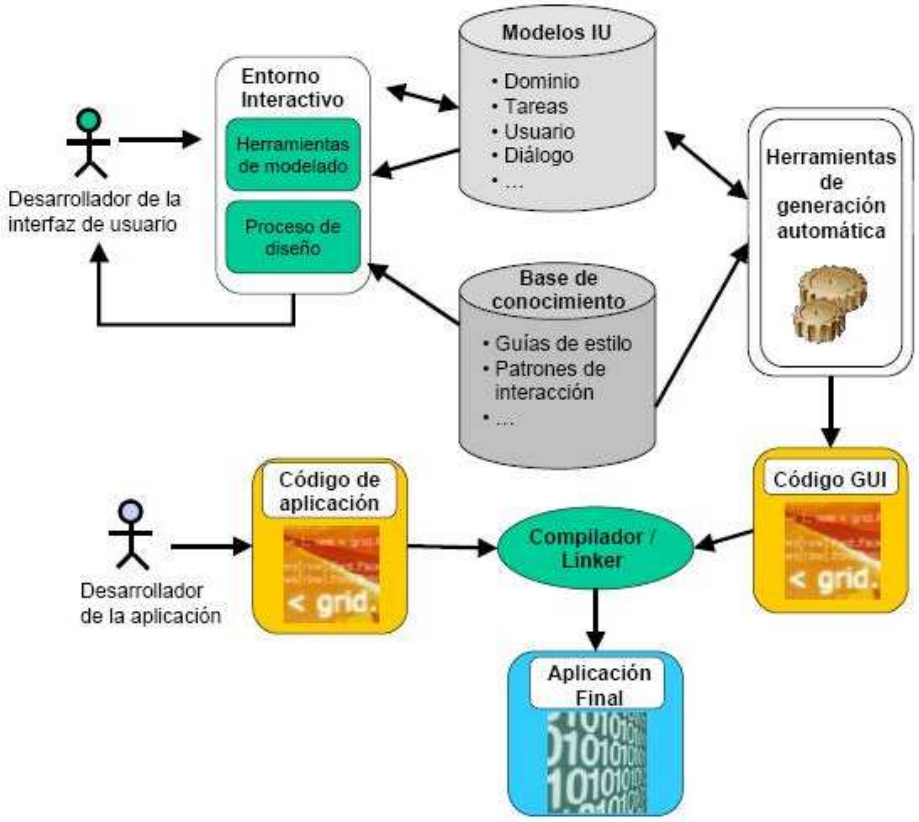


Figura 3.1: Arquitectura general de un entorno de desarrollo basado en modelos.

3.2.4. Ventajas y aportaciones

Las técnicas basadas en modelos proporcionan numerosos beneficios [Pin00]. Podemos mencionar los siguientes como más destacables:

- *Abstracción.* La descripción de la IU proporcionada por este tipo de herramientas es de un nivel de abstracción superior al de otro tipo de herramientas de desarrollo de IUs anteriores.
- *Automatización y productividad.* Proporcionan la infraestructura necesaria para la automatización de parte de las tareas de diseño y generación de IUs –sobretodo tareas repetitivas, más idóneas de automatizar que de resolver manualmente-, sin olvidar que el diseñador humano también toma parte en las decisiones más relevantes.
- *Homogeneidad y conformidad con estándares.* La resolución automatizada de ciertas tareas conlleva una mayor homogeneidad en los resultados, así como una mayor conformidad del código generado con los estándares considerados.

- *Sistematización.* Permiten diseñar e implementar las IUs de forma sistemática, ya que facilitan (a) el modelado de la IU en distintos niveles de abstracción; (b) el refinamiento incremental de los modelos; y (c) la reutilización de las especificaciones de diversos modelos.
- *Completitud.* Proporcionan un soporte completo para todo el ciclo de vida del sistema.
- *Implicación del usuario en el desarrollo.* Constituyen una metodología fácilmente integrable con las técnicas de *diseño centrado en el usuario*, lo que permite controlar en todo momento los parámetros preestablecidos de usabilidad.
- *Preservación de la usabilidad.* El establecimiento a priori de las propiedades de usabilidad a preservar, la integración de la experiencia adquirida por los desarrolladores en el propio marco de desarrollo, así como la supervisión humana favorecen la preservación de la usabilidad.
- *Reutilización.* Es posible reutilizar los modelos y diseños de la IU, así como también la reutilización del conocimiento de la IU para distintos desarrollos.
- *Minimización del coste de desarrollo.* La producción automática de las distintas *IUs específicas* a partir de una única *IU genérica* evita el sobreesfuerzo de desarrollar y mantener cada una de ellas por separado.
- *Otras características favorables* son: capacidad para destacar la información importante, asistencia en el manejo de la complejidad y validez para soportar métodos de desarrollo.

Como ventajas comunes a todas las herramientas de desarrollo de IUs podemos citar las siguientes: facilitan un prototipado rápido de la IU y su consecuente incorporación de cambios; alivian el proceso de creación de múltiples IUs; fomentan la consistencia y permiten que diversos especialistas se involucren en el proceso de diseño de la IU; la codificación es también más fácil y económica; facilitan la creación de métodos para diseñar la IU de manera sistemática; uso de un conocimiento de diseño explícitamente representado; la integración de herramientas de diseño que cubren todo el ciclo de vida.

3.2.5. Problemas y limitaciones

Por otra parte, estas aproximaciones también presentan actualmente ciertos problemas por resolver, así como considerables limitaciones, tal y como se expresa en [Sch96]. A continuación se relacionan las más destacables:

- *Complejidad de los modelos y sus notaciones.* Habitualmente no resulta sencillo aprender el funcionamiento y uso de los modelos y sus notaciones. No obstante, se espera que el desarrollo de herramientas de diseño visuales reduzca esta complejidad en gran medida. Además, las notables diferencias en rango, naturaleza y notación de los distintos modelos soportados dificulta su comparación y reutilización. Sería deseable disponer de alguna notación estándar. El compromiso que estas herramientas plantean es si realmente la especificación de la IU de partida es menor y más fácil de obtener que el propio programa a generar. Sólo en ese caso estará justificado recurrir a este enfoque. Esta problemática ha sido calificada como la *regla de Novak* [Mol04].

- *Falta de consenso.* No existe un consenso sobre cuáles son los modelos más adecuados para modelar una IU. Ni siquiera lo existe sobre cuáles son los aspectos de la IU que deben ser modelados.

- *Problema del mapeo entre modelos.* Este problema fue introducido por Puerta y Eisenstein en [PE99], y hace referencia a la dificultad para modelar las relaciones entre los modelos con el fin de dirigir y facilitar las sucesivas transformaciones de modelos abstractos en modelos más concretos. Se trata de un problema bien conocido en el desarrollo de IUs dirigido por transformación [PE99].

- *Problema de refinamiento post-edición.* Este problema se refiere al dilema que se presenta al tener que elegir entre regenerar nuevos diseños automáticamente a partir de un primer prototipo, o bien aplicar manualmente refinamientos incrementales a los diseños obtenidos, dado que no se puede sacar partido a una combinación de ambas estrategias [Pin00].

- *Limitación en la riqueza de la IU generada.* En muchas ocasiones las IUs obtenidas ofrecen un conjunto limitado de componentes de interacción, resultando notoriamente simples. Schlungbaum describe este problema diciendo que las IUs generadas son mayoritariamente *basadas en forma*[32] [Sch96].

- *Dificultad de integración de las distintas IUs con la aplicación subyacente.* La separación entre la IU y la funcionalidad de la aplicación es, en cierta medida, contraproducente, puesto que obliga a mantener dos especificaciones que comparten contenidos de modo separado, de manera que se hace necesario velar por mantenerlas sincronizadas. La integración de la IU obtenida con la aplicación subyacente constituye un problema reconocido en la literatura. Para ser más precisos, el método y mecanismos de integración a seguir para enlazar el modelo del dominio con la aplicación subyacente no está bien clarificado. Según Schlungbaum, la única forma

[32] IUs extremadamente simples, limitadas a un conjunto específico de objetos de interacción.

de integrar la IU con el desarrollo del sistema es el uso simultáneo del modelo de datos tanto en el desarrollo de la IU como en el de la aplicación [Sch96].

- *Falta de información semántica intrínseca en los modelos.* Pocos trabajos han enfocado sus esfuerzos en obtener una adaptación en base a la semántica de los modelos. A parte del trabajo mencionado anteriormente de Lemlouma y Layaïda para afrontar la adaptabilidad de contenidos (véase *sección 3.1.4.1*), únicamente en el marco de referencia unificado (el CAMELEON Reference Framework), que se presenta a continuación tienen en cuenta la genericidad de los modelos y su independencia del dominio (véase *sección 3.3.5*).

- *Falta de flexibilidad en la adaptación de contenidos y coherencia en entornos heterogéneos* [LL02b]. Este problema es identificado por Lemlouma y Layaïda, quienes presentan una infraestructura de adaptación con el propósito de ofrecer una adaptabilidad de contenidos universal, tal y como se ha mencionado anteriormente (véase *sección 3.1.3*). Su trabajo, no obstante, no está enmarcado dentro del enfoque basado en modelos.

- *Falta de anticipación a los cambios contextuales.* En general, los entornos de desarrollo basados en modelo son esencialmente estáticos, dejando sin resolver la anticipación a los cambios contextuales. Esto significa que las situaciones provocadas por cambios contextuales no están correctamente previstas, tal y como ocurre por ejemplo en ARTStudio (Adaptation by Reification and Translation) [The01] (véase *sección 3.3.3*). Sería deseable ofrecer una solución más dinámica.

- *Otras limitaciones de los enfoques actuales* son: el diseño se basa completamente en soluciones ad hoc, así como en la intuición del diseñador; las herramientas visuales no soportan mapeo entre las actividades lógicas y los elementos de la interfaz. Las herramientas basadas en UML están orientadas al diseño del sistema; falta de soporte para el diseño basado en tareas de usuario.

Definitivamente, este tipo de técnicas deben mejorar para afrontar el problema de la plasticidad. En nuestra opinión el conjunto de modelos que se consideran es bastante limitado. En general, sólo se consideran modelos de tareas, usuario y plataforma, dejando sin modelar los aspectos contextuales, salvo en casos muy puntuales. Además, el hecho de que sean esencialmente estáticas también limita mucho su idoneidad. A pesar de que estos aspectos empiezan a integrarse en los trabajos de Calvary et al. [CCT01a], sólo se llevan a la práctica en Probe, que se encuentra en fase experimental (véase *Capítulo 1*; *sección 1.2.2*). En definitiva, la anticipación a los cambios contextuales permanece aún sin

resolver. En esta tesis se defiende que un completo modelado contextual (conjunción de los modelos de plataforma, entorno, usuario, e incluso alguno más especíco, dependiendo del tipo de sistema debe ser también considerado y apropiadamente relacionado con el resto de modelos, participando activamente en el proceso de construcción de IUs plásticas.

No obstante, la intervención de este modelo debe estar respaldada por un adecuado mecanismo de propagación de cambios contextuales y un sistema de realimentación consecuente que garantice que los cambios sufridos durante el uso del sistema repercuten también en la operativa llevada a cabo en la generación de IUs multi-contextuales, garantizando su transmisión y puesta en escena. Precisamente, la ausencia de un mecanismo para propagar cambios y actualizar los modelos es una de las limitaciones detectadas en la literatura [Sze96]. Entre las contadas herramientas que cuidan este aspecto podemos nombrar MASTERMIND [SSC+96], que únicamente permite que los cambios realizados en tiempo de diseño en un modelo se propaguen a los otros modelos. Otro ejemplo es la herramienta Teallach [GFP+98], que aplica un enfoque de modelos compartidos para atacar este problema.

3.2.6. Los modelos en MB-UIDE

Actualmente no existe un estándar que defina cuáles son los modelos que debería tener un entorno de desarrollo basado en modelos. Muchas de las técnicas basadas en modelos existentes utilizan un conjunto distinto de modelos declarativos, cuya elección depende en cada caso de qué factores se consideran relevantes, y que por tanto influirán en la configuración de la IU. En parte, este problema es debido a las distintas disciplinas involucradas en la definición de los MB-UIDE. Así, nos encontramos iniciativas afines a la IPO, como son las de [Pue97] y [VB93], otras más afines a la Ingeniería del Sofware, como [LGR00] y [PIP+97], y finalmente otras que proceden del mundo de la hipermedia, como [GCP01] o [Koc00]. Sin embargo, sí que existen un conjunto de modelos comunes que aparecen en prácticamente todas las aproximaciones. Estos son los modelos de dominio, tareas, diálogo, usuario y presentación. En particular, el enfoque orientado a tareas para el desarrollo de IUs generalmente adoptado en las herramientas basadas en modelos proviene de un área de investigación sólida en el campo de IPO, y se basa en explotar el conocimiento capturado durante una fase temprana del análisis de requisitos [PLL04], [Pat99].

Veamos en qué consisten estos modelos tan ampliamente utilizados.

3.2.6.1. El Modelo del Dominio

El *modelo de dominio* proporciona una representación de los objetos sobre los que actúan las tareas capturadas en el *modelo de tareas*. Proporciona información acerca del tipo, atributos y operaciones de los objetos del dominio identificados.

La especificación del modelo del dominio va muy ligada al desarrollo de la parte funcional de la aplicación, que habitualmente incluye también un *modelo del dominio*. Este factor puede producir duplicidad de información, con el consecuente peligro de aparición de incoherencias. Además, repercute negativamente en la integración de la IU con la aplicación subyacente, problema presentado anteriormente.

Algunas aproximaciones utilizan para la representación del *modelo del dominio* diagramas entidad/relación [Che76], como por ejemplo GENIUS [JWZ93], mientras que otras utilizan técnicas de orientación a objetos -diagramas de clases principalmente-, como por ejemplo MECANO [Pue96] o IDEAS [LGR00]. Igual que para los otros modelos, no existe un método estándar para la representación del *modelo del dominio*.

Entre las notaciones utilizadas para especificar este modelo se puede encontrar desde referencias informales a los objetos del dominio hasta paradigmas estructurados tales como UML o diagramas de entidad/relación.

3.2.6.2. El Modelo de Tareas

El *modelo de tareas* constituye el eje central de los métodos basados en modelos. Consiste en una representación estructurada de las tareas que un usuario puede realizar a través de la IU. Se entiende por tarea todo aquello que contribuye a la consecución de un objetivo concreto, y que generalmente contribuye a la modificación del estado del modelo del dominio. Las tareas se descomponen en acciones atómicas que representan los pasos necesarios para alcanzar los objetivos de la tarea. Dentro de los datos capturados en este modelo también se recogen los requisitos no funcionales de dichas tareas, como por ejemplo el de tiempo de respuesta, así como la especificación de las restricciones a respetar en el orden de realización de las mismas.

Para la especificación del *modelo de tareas* se han utilizado distintas aproximaciones, entre las que destacan los métodos textuales basados en el análisis cognitivo como GOMS (Goals, Operators, Methods, Selection rules) [CMN83], [NC85], o los métodos basados en formalismos como el ConcurTaskTrees [PMM97], presentado por Paternò. Una técnica muy utilizada para la captura de los requisitos de las distintas tareas que se realizarán con la IU es la de los diagramas de casos de uso [BJR99].

Las ventajas que ofrece disponer de una especificación de tareas se pueden resumir en: (1) una mejor captura de requisitos; (2) un diseño de IU detallado y consistente; y (3) una mejor integración con las situaciones de la vida real [DFAB03].

3.2.6.3. El Modelo del Diálogo

El *modelo de diálogo* describe las posibles "conversaciones" entre la IU y el usuario definiendo estados de la IU y transiciones entre estados, poniendo de manifiesto qué transiciones son posibles y qué no entre los mismos, así como la secuenciación de la información. Representa cuándo el usuario puede introducir datos, seleccionar, o bien cuándo se muestran los datos, y a la inversa, cuándo el ordenador puede requerir esos datos, proporcionando una descripción del comportamiento dinámico. En general, el modelo de diálogo representa una secuencia de entradas y salidas que permitirá establecer el estilo de navegación.

Para la representación del modelo de diálogo se utilizan tanto técnicas textuales como visuales. Entre las más utilizadas están las distintas variaciones de los diagramas de transición, las redes de Petri [BP90], los diagramas de secuencia o diagramas de estados de UML [Har87], [LGR00], la notación State Transition Networks [Was85], Dialogue Graphs [SE96b], Window Transitions [VLF03], Hierarchic Interaction graph Templates (HIT) [Sch94], etc.

3.2.6.4. El Modelo del Usuario

El *modelo de usuario* captura las características y requisitos individuales de cada usuario o grupo de usuarios [Fis01], tales como el nivel de conocimiento, preferencias, metas, estado, necesidades, etc. Habitualmente, cada uno de los tipos de usuario que interactuarán con la aplicación se denomina *rol*. El objetivo de este modelo es ofrecer una IU que se ajuste a las características y requisitos de cada usuario, tratando de personalizar la funcionalidad o apariencia de la IU.

La adaptación de la IU al usuario puede realizarse en tiempo de diseño o en tiempo de ejecución, siendo ésta una de las distinciones entre *sistemas adaptables* o *adaptativos*, tal y como se presenta en el *Capítulo 2* (véase *Capítulo 2*; *sección 2.1.2.*). En tiempo de diseño se puede definir, por ejemplo, cuáles serán las tareas disponibles para cada tipo de usuario. La adaptación en tiempo de ejecución, sin embargo, requiere un modelo más completo [HBH+98], que además puede ser evolutivo como consecuencia de la inferencia del patrón de comportamiento del usuario u otras observaciones llevadas a cabo durante el proceso de interacción. Hasta la fecha, los modelos de usuario introducidos dentro del

desarrollo de IUs basadas en modelos han sido bastante reducidos e insuficientes para la adaptación de la IU en tiempo de ejecución.

Por otra parte, las características del usuario contenidas en el modelo de usuario se suelen clasificar en *dependientes de la aplicación* e *independientes de la aplicación* [Hol90]. Las características independientes de la aplicación podrán ser reutilizadas de una aplicación a otra para el mismo usuario, mientras que las dependientes de la aplicación deberán ser elaboradas en cada caso.

Cabe decir que no existen notaciones formales para especificar el modelo de usuario y la mayoría de sistemas ni siquiera describen cómo es utilizado a lo largo del proceso. Por ello se puede afirmar que se trata del modelo menos especificado [GBM+99].

3.2.6.5. El Modelo de Presentación

Se acostumbra a denominar *modelo de presentación* a una descripción más o menos abstracta de la IU final con la que el usuario interactuará, resultado del proceso de explotación de los modelos anteriores. Este modelo representa los componentes de interacción que dan forma a la IU, las características de disposición, su apariencia y dependencias visuales entre componentes.

En la mayoría de entornos existen dos niveles distintos pero complementarios de modelos de presentación. Por una parte se construye un modelo abstracto, el cual constituye una vista abstracta de una IU genérica que representa el modelo de tareas subyacente. Se trata de una descripción de la IU en función de los llamados *objetos abstractos de interacción*, y por otra parte se construye un modelo de presentación concreto, que se compondrá de *objetos concretos de interacción* [VB93].

Esta separación en nivel abstracto y concreto del modelo de presentación permite una generación de la IU no sólo para distintas plataformas, sino también para distintas condiciones del entorno o también para distintas características y habilidades del usuario, a partir de una misma descripción abstracta de la IU. Un proceso de transformación automatizado seleccionará los *objetos concretos de interacción* correspondientes a los *objetos abstractos de interacción*, siguiendo un determinado modelo o conjunto de reglas de transformación. Este proceso suele ir dirigido por un conjunto de heurísticas, guías de estilo o patrones de interacción que recogen la experiencia adquirida por los desarrolladores de IUs.

Han sido propuestos en la literatura distintos conjuntos de *objetos abstractos de interacción* para afrontar el diseño abstracto de la IU como son [VB93], [Pin02] y [LGR00], entre otros.

3.3. Desarrollo cronológico hacia un Marco de Referencia Unificado

Con el objetivo de desarrollar un instrumento de referencia no sólo para caracterizar, clarificar y contrastar el trabajo existente en el dominio de plasticidad, sino también para identificar nuevas oportunidades inexploradas en el desarrollo de IUs plásticas, se inicia la construcción de un marco conceptual de referencia con el fin de consensuar un marco de trabajo común en el desarrollo de IUs multi-contextuales. A continuación se detallan cada uno de los pasos evolutivos llevados a cabo hacia la completitud de este marco donde, tal y como se expone a continuación, se van ampliando paulatinamente tanto el número de herramientas que es capaz de representar, como los requisitos de adaptación y plasticidad involucrados.

3.3.1. Sentando las primeras bases

Juntamente con el acuñamiento del término plasticidad en [TC99], se presentan las bases sobre las cuales se inicia la construcción del primer marco de referencia para soportar el desarrollo de IUs plásticas. Basándose en el *enfoque basado en modelos*, estos autores estructuran el espacio del problema proporcionando los cimientos para un espacio de solución utilizando especificaciones de alto nivel de abstracción y abriendo el camino hacia la generación automática o semi-automática de IUs plásticas. Además, ilustran la factibilidad del enfoque con dos casos de estudio: una agenda electrónica portátil y un espacio multimedia que representa gráficamente el nivel de actividad de los miembros de un laboratorio, el cual se adapta a diferentes tamaños de pantalla física entre ordenadores de sobremesa estándar y que denominan MMS (MultiMedia Space).

3.3.1.1. Presentación del Marco Conceptual

Las bases del marco conceptual que se va definiendo en esta primera contribución aportan un valor añadido a los modelos y métodos de diseño existentes hasta entonces en IPO (por ejemplo MUSE [DL89], ADEPT [JWMP93] o MAD [SP90] en dos aspectos: (1) reutiliza y mejora los modelos existentes hasta el momento en el campo del diseño y desarrollo de IUs con el fin de satisfacer los requisitos impuestos por la plasticidad y poder representar el contexto de uso; y (2) introduce explícitamente nuevos modelos y heurísticas que hasta entonces habían sido ignorados para representar el contexto de uso.

La figura 3.2 (pág. siguiente) muestra las siete componentes de este primer esbozo del marco conceptual.

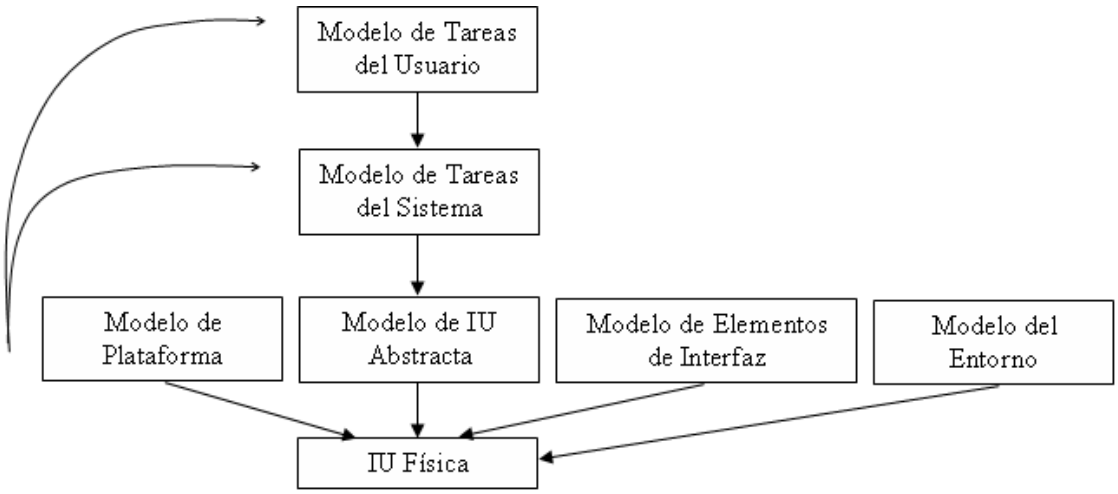


Figura 3.2: Estructura de derivación de modelos para la generación de IUs plásticas.

A continuación se presenta una breve descripción de cada una de ellas, desde la concepción del sistema hasta su derivación en una IU concreta. El *Modelo de Tareas del Usuario* es una trascripción formal o semi-formal de las actividades del mundo real. El *Modelo de Tareas del Sistema* describe cómo esas tareas podrían ser llevadas a cabo a través del sistema que se está diseñando, esto es, constituye una implementación del *Modelo de Tareas del Usuario*. A continuación, el *Modelo de IU Abstracta* es una expresión canónica de una representación abstracta de los conceptos del dominio de acuerdo al *Modelo de Tareas del Sistema*. Se trata de una representación independiente del *elemento de interfaz* (del término inglés *interactor*[33]). El *Modelo de Elementos de Interfaz* describe los elementos de interfaz disponibles para la representación de una IU particular. El *Modelo de Plataforma* propuesto en este esquema tan sólo describe las características físicas de las plataformas objetivo, en términos de recursos hardware. El *Modelo del Entorno* especifica el contexto de uso que para estos autores abarca los objetos, personas y eventos que rodean la tarea que se está realizando y que pueden tener un impacto en el sistema y en el comportamiento del usuario. Por último, la *IU Física* resulta de la resolución y propagación de restricciones expresadas en el *Modelo de IU Abstracta* y los *Modelos de Elementos de Interfaz, de Plataforma* y *del Entorno*.

Como se observa en la figura 3.2, las flechas indican las dependencias entre los modelos en el proceso de derivación de una IU particular. Aunque se trata de un proceso básicamente top-down, tal y como aparece reflejado, los modelos relacionados con el mundo real (el *Modelo de Plataforma*, el *Modelo de Elementos de Interfaz* y el *Modelo del Entorno*), los cuales están situados en el tercer nivel, pueden tener un impacto en los modelos de mayor nivel de abstracción, abriendo el camino hacia otros recorridos en el proceso de

[33]Abstracción que modela los conceptos que es capaz de representar y las tareas de usuario que soporta.

derivación de las IUs plásticas. Esto es lo que se representa con las flechas en sentido ascendente. Ciertamente, en la realización de la tarea de lectura del correo electrónico, las necesidades del usuario pueden variar dependiendo de si se realiza a través de un teléfono móvil o de un ordenador de sobremesa.

Por supuesto, para que este marco sea operativo, los modelos implicados deben ser estructurados y formalizados. En ese sentido, los autores proporcionan también un conjunto de requisitos y heurísticas que se presentan a continuación.

3.3.1.2. Requisitos, Heurísticas y Recomendaciones

Estas heurísticas son fruto del análisis de los métodos de diseño de IUs existentes hasta la fecha, en los que la plasticidad no había sido considerada. En lo sucesivo, constituyen los fundamentos sobre los que construir cualquier método, modelo de proceso o herramienta de plasticidad.

En primer lugar, tal y como Thevenin y Coutaz expresan en [TC99], en la literatura hasta la fecha no se especificaba, salvo excepciones, la importancia relativa de los conceptos del dominio, ni tampoco su ámbito. Basándose en esa observación estos autores dan la siguiente recomendación:

> *"los conceptos del dominio involucrados en una tarea deberían ser clasificados de acuerdo tanto a su nivel de importancia en el dominio de tareas como a su grado de implicación para cada punto de interacción particular" [TC99].*

Según estos autores, estos aspectos pueden ayudar en el proceso de representación utilizando heurísticas generales. En concreto proponen las siguientes:

Heurística 3.1 *El Modelo de Tareas del Sistema incluye tareas dependientes del dominio y tareas articulatorias, esto es, tareas independientes del dominio que, como tales, no aparecen en el Modelo de Tareas del Usuario ni tampoco afectan a los conceptos del dominio. El scroll y la navegación constituyen ejemplos de tareas articulatorias típicas. Por este motivo establecen la siguiente recomendación:*

> *"Las tareas articulatorias genéricas deberían ser identificadas de manera sistemática e insertadas apropiadamente en el Modelo de Tareas del Sistema" [TC99].*

Heurística 3.2 *Aunque se han desarrollado múltiples representaciones formales para especificar elementos de interfaz, los cuales se consideran adecuados para un contexto de uso particular, ninguno de ellos dirige el problema de la plasticidad explícitamente. Esta observación les conduce a formular la siguiente recomendación al respecto:*

> *"Los elementos de interfaz deberían especificar los tipos de datos abstractos que son capaces de manejar para una IU particular. Deberían ser capaces de evaluar su conveniencia para un contexto de uso determinado, así como también su coste de representación" [TC99].*

Por lo que concierne a los tipos de datos abstractos, éstos proporcionan la información necesaria para realizar el mapeo entre un concepto del dominio y un conjunto de *elementos de interfaz* candidatos. Hasta entonces, sólo las técnicas de modelado y las reglas de mapeo de Vanderdonckt [Van95] para IUs gráficas, así como las de Sutcliffe [Sut97] y Bernsen [Ber94] para IUs multimedia dirigían este problema considerando una interacción multimodal.

Es importante remarcar que la capacidad de un *elemento de interfaz* para evaluar su conveniencia permite guiar el proceso de selección en la representación de conceptos cuantificando la continuidad o degradación de la usabilidad del sistema. Este aspecto se equipara con los objetivos de plasticidad. Por otro lado, el *coste de representación* de un *elemento de interfaz* expresa la cantidad de recursos físicos necesitados para su instanciación. Este coste soporta también la selección del conjunto final de *elementos de interfaz*.

Heurística 3.3 *Para alcanzar los objetivos de la plasticidad se requiere la expresión explícita de las plataformas objetivo en términos de los recursos físicos cuantificados. Debe incluir los dispositivos de interacción disponibles, capacidad de cómputo (memoria y capacidad de procesamiento principalmente) y las capacidades de comunicación (ancho de banda y canales de comunicación). Hasta entonces se había considerado implícitamente el ordenador de sobremesa como Modelo de Plataforma.*

Heurística 3.4 *La noción de Unidad de Presentación utilizada en TRIDENT [Van95] sienta las bases de la representación canónica del Modelo de IU Abstracta.*

La *Unidad de Presentación* es una estructura jerárquica que modela contenedores de información y unidades elementales que corresponden a los conceptos del dominio en el nivel apropiado de granularidad. Constituyen la parte encargada de llevar a cabo la entrada y visualización de los datos de cualquier sub-tarea de una tarea interactiva. Además, muestran un comportamiento y están relacionadas con otras *Unidades de Presentación* a través de relaciones de navegación que reflejan la ordenación de las tareas.

En consecuencia, se distinguen dos tipos de reglas de mapeo: (1) para la correspondencia entre *Unidades de Presentación* y *elementos de interfaz*; y (2) para la correspondencia entre *elementos de interfaz* y los recursos físicos proporcionados por el sistema.

3.3.2. Primer Modelo de Proceso de Desarrollo de IUs Plásticas

El siguiente paso es el que aportan Calvary, Coutaz y Thevenin en [CCT00]. Se presenta un nuevo modelo de proceso que suple la ausencia de notaciones apropiadas para hacer referencia al contexto de uso. Como modelo de proceso estructura el desarrollo de sistemas interactivos plásticos. Constituye una primera versión de lo que es hoy en día el *Marco de Referencia Unificado* para IUs multi-contextuales.

La figura 3.3 muestra los modelos involucrados en el proceso. El *Modelo de Conceptos* corresponde al *Modelo de Tareas del Usuario* del esquema anterior; el *Modelo de Tareas* con el *Modelo de Tareas del Sistema*; la *IU Abstracta* con el *Modelo de IU Abstracta* y, por último, el *Sistema Ejecutable para un contexto de uso* a la *IU Física* del esquema anterior. En particular, el *Modelo de Conceptos* constituye lo que actualmente se conoce universalmente como *modelo del dominio*. Además, como se observa en la figura, aparece un nuevo modelo: el *Modelo de Evolución*, el cual especifica el cambio de estado dentro de un contexto, así como las condiciones necesarias par alcanzar y abandonar un contexto particular. Según estos autores, el *Modelo de Elementos de Interfaz* describe *"los elementos de interfaz multi-modal sensibles al recurso, disponibles para producir una IU concreta"* [CCT00].

En concreto, la figura muestra el proceso cuando es aplicado a dos contextos distintos: *contexto 1* y *contexto 2*. Los modelos de las dos columnas centrales son denominados *modelos transitorios*, para diferenciarlos de los demás, los *modelos iniciales* –aquellos que son especificados manualmente por el desarrollador. Se trata de modelos intermedios necesarios para la producción de la IU ejecutable final, los cuales requieren de una referencia continua a los modelos iniciales para llevar a cabo todo el proceso de desarrollo. La *IU Concreta*, que no se contemplaba en el esquema anterior [TC99], convierte una *IU Abstracta* en una expresión dependiente de los *elementos de interfaz*. A pesar de que la *IU Concreta* explicita la apariencia final de la IU final, todavía consiste en un prototipo que sólo es operativo en el entorno de desarrollo utilizado. Así, por ejemplo, en la herramienta SEGUIA -presentada en la sección 3.5- [VB99], una IU concreta consiste en una jerarquía de *Objetos Concretos de Interacción* (OCIs) que resulta de una transformación de *Objetos Abstractos de Interacción* (OAIs) [VB93]. Estos OCIs pueden ser visualizados en el propio sistema, pero esa IU no constituye todavía una IU operativa. Por ello es necesario un nuevo paso de derivación para obtener el sistema ejecutable para un contexto de uso concreto –la IU final-, tal y como se plasma en la parte inferior de la figura 3.3.

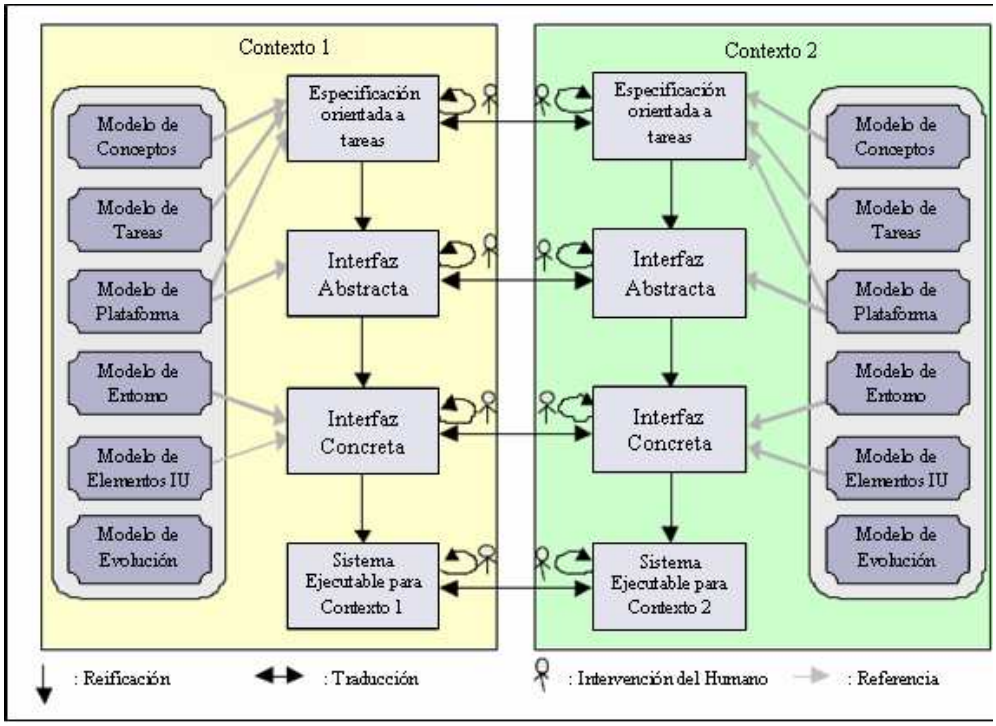


Figura 3.3: Proceso de desarrollo de referencia para soportar sistemas interactivos plásticos.

Tal y como se observa en la figura, el proceso es definido como una combinación de dos operaciones: *reificación*[34] vertical y *traducción*[35] horizontal, las cuales pueden ser llevadas a cabo tanto manualmente por parte del experto humano, como automáticamente a través de la herramienta. En particular, se recurre a la acción manual cuando las herramientas no pueden preservar los criterios de usabilidad establecidos. La reificación vertical cubre el proceso de derivación de los modelos (de los modelos más abstractos a una implementación concreta). Las derivaciones horizontales corresponden a traducciones entre modelos en el mismo nivel de reificación. La combinación de operaciones de reificación y traducción puede evitar la producción de modelos intermedios.

Este modelo conceptual puede ser instanciado de muchas maneras. La situación ideal consiste en aplicar reificación hasta el último nivel, y entonces aplicar directamente la traducción a otro contexto de uso distinto. Este proceso conllevaría un esfuerzo mínimo para mantener la consistencia entre las IUs producidas. En la figura 3.4 (b) se refleja este caso. El caso reflejado en (a) corresponde a la práctica actual, es decir, la obtención de IUs ejecutables de manera paralela. Esta estrategia requiere un gran esfuerzo por man-

[34]Transformación de un modelo para un contexto dado a un modelo más concreto para el mismo contexto.
[35]Transformación de un modelo para un contexto dado al mismo tipo de modelo para otro contexto, como operación que soporta los cambios contextuales.

tener la consistencia de manera manual entre las distintas versiones. Una instanciación común del marco conceptual consistiría en una combinación de operaciones de reificación y traducción, tal y como se muestra en (c).

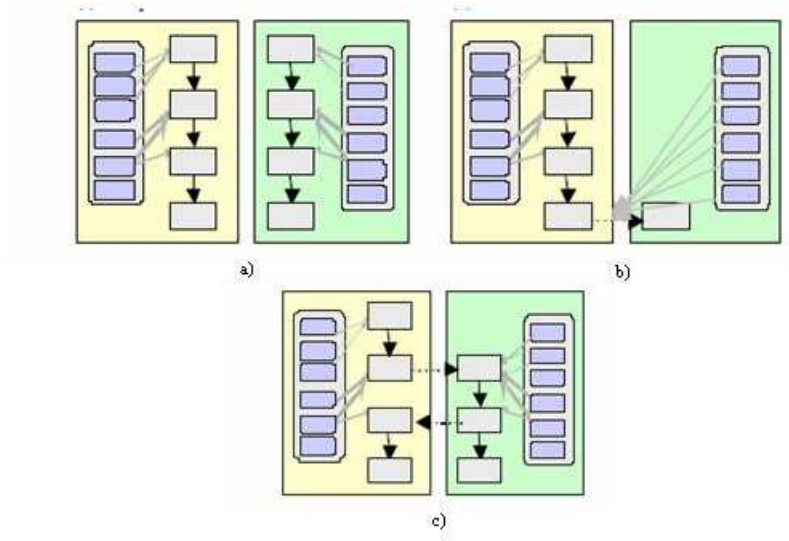


Figura 3.4: Tres posibles instanciaciones del marco de referencia: (a) práctica actual; (b) caso ideal y (c) caso común.

Este marco de referencia, aunque incompleto, proporciona ya una estructura de referencia para afrontar el problema objeto de estudio.

3.3.3. ARTStudio: una instanciación del primer Modelo de Proceso

ARTStudio (Adaptation by Reification and Translation) [The01] es la primera herramienta generadora de IUs multi-contextuales que soporta el marco de referencia descrito anteriormente. En realidad se trata de una implementación incompleta del mismo puesto que, además de que sólo actúa la operación de reificación vertical -la traducción no está soportada-, los *Modelos de Entorno* y *Evolución*, los cuales forman parte ya de la infraestructura de tiempo de ejecución, no están implementados. Además, en su estado actual el *Modelo de Plataforma* es bastante primitivo. La figura 3.5 muestra su cobertura funcional.

De acuerdo al marco de referencia, la *Especificación orientada a tareas* se obtiene a través de los *Modelos de Conceptos* y *de Tareas*, la cual da lugar a la generación automática de la *IU Abstracta*. A partir de allí, el *Modelo de Plataforma* y el *Modelo de Elementos de interfaz* entran en funcionamiento para generar automáticamente la *IU Concreta*. A

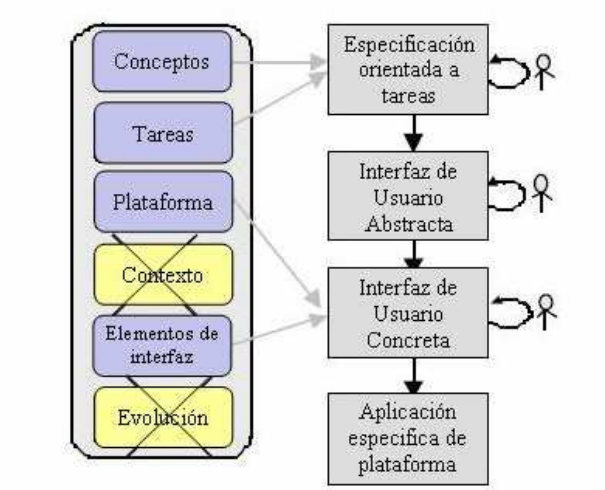


Figura 3.5: Cobertura funcional de ARTStudio en relación al marco de referencia.

cada paso del proceso de reificación el diseñador tiene la posibilidad de reajustar las descripciones generadas por el sistema.

ARTStudio está implementado en Java y utiliza CLIPS, un lenguaje basado en reglas para la generación de la *IU Concreta*. Tanto los *modelos iniciales* como los *modelos transitorios* se soportan en ficheros XML. Las IUs ejecutables finales se expresan en Java. En su versión actual ARTStudio sólo resuelve variaciones del tamaño de pantalla. A continuación se describen los modelos que soporta.

- **Modelo de Tareas**. Consiste en una estructura ConcurTaskTree [PMM97].
- **Modelo de Conceptos**. Los conceptos son modelados en ARTStudio utilizando especificaciones UML estándar.
- **IU Abstracta**. Es modelada como un conjunto estructurado de espacios de trabajo isomórficos y relacionados entre sí. Existe una correspondencia uno a uno entre un espacio de trabajo y una tarea. Cada espacio de trabajo incluye tantos sub-espacios como sub-tareas contenga la tarea abstracta. Las relaciones entre los espacios de trabajo son inferidos de las relaciones entre tareas expresadas en los *Modelos de Conceptos* y *Tareas*.
- **Modelo de Plataforma**. Descripción UML que captura las siguientes características: el tamaño y profundidad[36] de la pantalla y el lenguaje de programación soportado por la plataforma. En su versión actual, este modelo es muy primitivo, aunque suficiente para demostrar los conceptos clave del marco de referencia.

[36]Número de colores soportados.

- **Modelo de Elementos de Interfaz**. Presenta la siguiente solución técnica:
  - *capacidad representacional*: un *elemento de interfaz* puede servir como mecanismo para transitar entre espacios de trabajo (por ejemplo un botón o una pestaña), o bien para representar conceptos del dominio. En este caso, el *Modelo de Elementos de Interfaz* incluye la especificación de los tipos de datos que es capaz de representar, de acuerdo a la *Heurística 3.2* anterior.
  - *capacidad de interacción*: las tareas que el *elemento de interfaz* es capaz de de soportar, como son la especificación, selección, navegación, etc.
  - *coste de uso*: mide tanto los recursos del sistema como los recursos humanos que el *elemento de interfaz* requiere, de acuerdo a la *Heurística 3.2* anterior.
- **IU Concreta**. Consiste en un conjunto de funciones de mapeo entre espacios de trabajo y superficies de la pantalla (ventanas o *canvas*), entre conceptos y *elementos de interfaz* y entre el esquema de navegación y los elementos de interfaz de navegación. En concreto, el mapeo de los conceptos en *elementos de interfaz* se basa en un sistema de resolución de restricciones.

3.3.4. Un Marco de Referencia Revisado

Previo a la definición del marco de referencia unificado para el desarrollo de IUs multi-contextuales definido bajo el proyecto CAMELEON (el CAMELEON Reference Framework), se llevan a cabo dos pasos más de refinamiento en el proceso de desarrollo de IUs plásticas anterior.

3.3.4.1. Refinamiento del diseño y cobertura en tiempo de ejecución

El marco conceptual anterior, propuesto en [CCT00], permite expresar varios enfoques para la producción de IUs multi-contextuales, aunque presenta una serie de limitaciones:

- el marco de referencia sólo soporta un enfoque top-down bajo un esquema rígido que no admite ningún paso del tipo bottom-up o combinación de ambos.
- no soporta enfoques de *ingeniería inversa*[37], esto es, que una nueva IU pueda obtenerse mediante la transformación de una IU ya existente de más bajo nivel de abstracción. Esto implica añadir una nueva operación: la operación de *abstracción*, representada como una flecha de abajo a arriba, a aplicar en la *IU Abstracta* o *IU Concreta*.

[37]Proceso que permite obtener una nueva IU mediante la transformación de una IU ya existente.

- El marco inicial asume que los únicos puntos de partida en el proceso son los modelos situados en el nivel superior o inferior del marco. Esto es bastante limitado y, en efecto, existen técnicas que no lo cumplen.
- El punto de partida es siempre único en el marco conceptual, lo cual no siempre se da. Así por ejemplo, Teallach [GBM+99], una herramienta basada en modelos, constituye un ejemplo representativo que considera varios puntos de partida, dando lugar a múltiples configuraciones que combinan esquemas de top-down y de bottom-up.
- Por último, el marco conceptual no soporta IUs cuya necesidad surja en tiempo de ejecución. Las IUs susceptibles de generar son todas predefinidas en fase de diseño.

La nueva versión revisada da solución a todas estas limitaciones detectadas [CCT+02b], dando lugar a un marco mucho más general que contempla todos los enfoques basados en modelos existentes. Además de introducir todos estos puntos para refinar la fase de diseño, extiende su cobertura al tiempo de ejecución, de acuerdo a los mecanismos de software y al proceso propuesto en [CCT01a]. A continuación se presentan ambos aspectos por separado.

Fase de diseño

Las variaciones introducidas son las siguientes:

- la noción de *punto de entrada*, como un indicador del nivel de reificación en el cual puede empezar el proceso de desarrollo. Se prevé un punto de entrada a cada nivel de reificación.
- un proceso de ingeniería inversa que permite inferir modelos abstractos a partir de modelos más concretos. Esto permite, por ejemplo, obtener un prototipo de *IU Concreta* sin haber producido previamente la *Especificación orientada a tareas*. Estas especificaciones pueden entonces obtenerse a partir de los prototipos por abstracción. Por lo tanto, es factible de producirse bucles entre modelos.
- Se introduce un operador de cruce en el nivel de traducción, el cual permite tanto la traducción a otro contexto de uso como un cambio en el nivel de reificación en un solo paso. Se puede entender como un acceso directo que permite evitar la producción de modelos intermedios.

Fase de ejecución

Tal y como se propone en [CCT00], el mecanismo de software que soporta cambios contextuales en tiempo de ejecución se estructura en un proceso en tres pasos: (1) *reconocimiento de la situación*, que incluye tanto la detección del contexto de uso a través de sensores como la consciencia de la magnitud e implicación del cambio contextual; (2) *computación de una reacción*, que comporta la identificación de las posibles reacciones y la selección de la candidata de acuerdo a un *coste de migración*[38] aceptable (véase *Capítulo 2*; *sección 2.1.5*); y (3) *ejecución de la reacción*. La ejecución de la reacción consiste en (1) un *prólogo* o preparación de la reacción (por ejemplo, completar, suspender o abortar la tarea actual antes de presentar la nueva IU, que si no está ya producida se genera en tiempo de ejecución); (2) la propia *ejecución de la reacción*, que produce la conmutación a una nueva versión (una nueva presentación, secuencia de diálogo o ejecución de una tarea específica); y (3) el *epílogo*, que finaliza la reacción, incluyendo la restauración del contexto de ejecución. Por ejemplo, ante una situación de saturación de la memoria de un dispositivo compacto, los pasos a realizar serían: (1) salvar el estado de las aplicaciones en ejecución y el estado de los documentos de trabajo (*prólogo*); (2) reiniciar el sistema (*ejecución de la reacción*); y (3) abrir de nuevo los documentos de trabajo (*epílogo*).

Cada uno de estos pasos es llevado a cabo por medio del sistema, el usuario o bien una combinación de ambos. En el caso de ser de manera automatizada, el mecanismo que actúa es lo que estos autores denominan un *supervisor del contexto*, que automáticamente detecta cambios del contexto y que está acoplado de forma transparente a la aplicación.

La figura 3.6 muestra ambas fases del modelo de referencia revisado.

En concreto, estos aspectos relativos al soporte en tiempo de ejecución han sido puestos en práctica en la herramienta *Probe* [Cal98], tal y como se mencionó en el *Capítulo 1* (véase *Capítulo 1*; *sección 1.2.2*). En *Probe* las reacciones son especificadas en un *Modelo de Evolución* (véase figura 3.6), y la detección de la desviación a partir de una situación dada se realiza automáticamente a través del software de *Probe*. La figura 3.7 2 (2 pág. adelante) ilustra el mecanismo de tiempo de ejecución en *Probe*.

El *Modelo de Evolución* especifica la reacción a aplicar ante un cambio del contexto. Otro aspecto a destacar es que este mecanismo soporta la anticipación a cambios contextuales posteriores, es decir, la ejecución de una reacción adecuada al contexto de uso que se prevé alcanzar en un futuro cercano, basándose en un historial que registra las transiciones entre versiones de la IU previamente realizadas.

[38] Este coste se mide como una combinación de criterios seleccionados en las fases tempranas del proceso de desarrollo, tal y como se indica en [CCT01a].

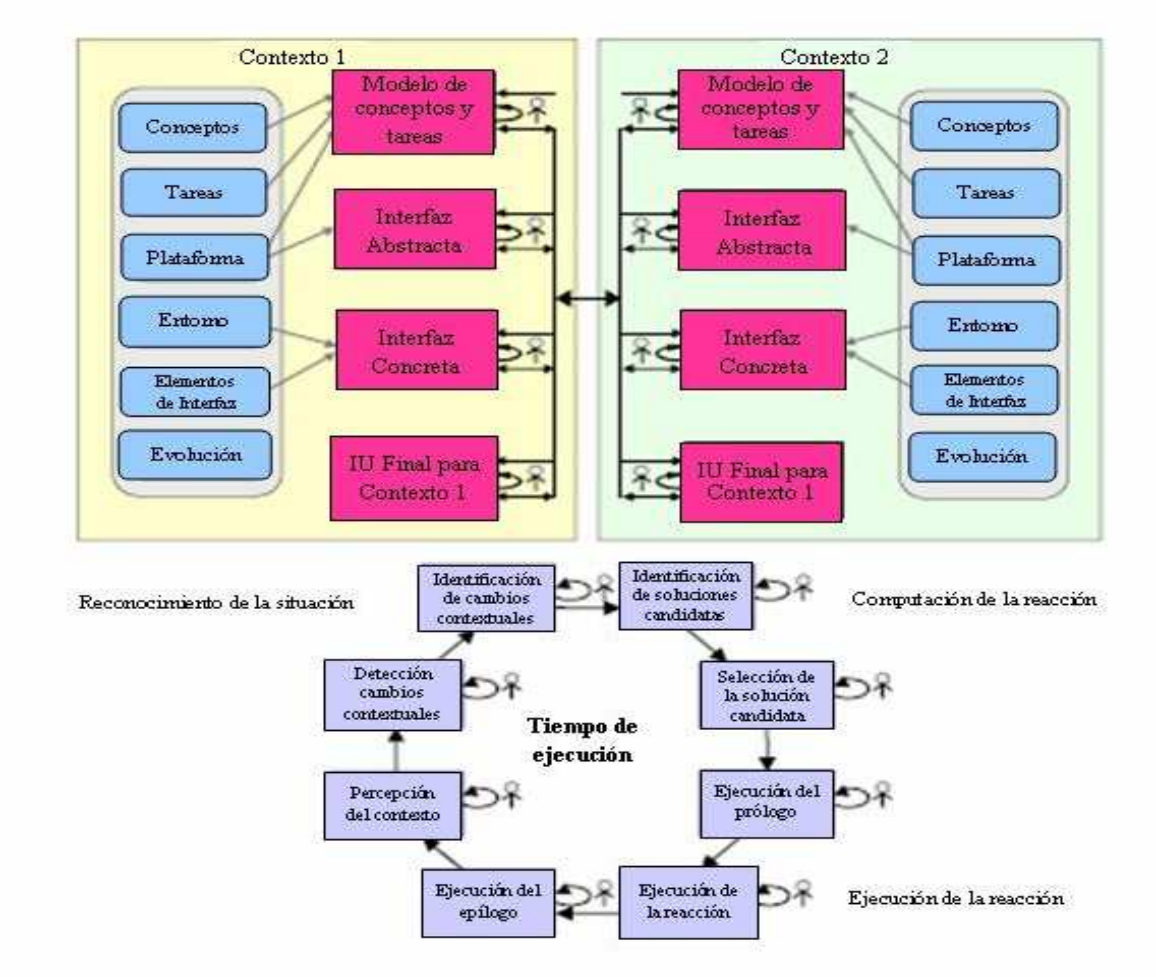


Figura 3.6: Marco de referencia resultante del primer refinamiento.

Este trabajo deja abiertos gran cantidad de aspectos, como son la inclusión de recomendaciones y heurísticas para clasificar los entornos y las plataformas. Además, cabe mencionar que estas ideas deben ser desplegadas de manera sistemática para dirigir la transición entre contextos de manera transparente.

3.3.4.2. Un paso más de refinamiento

De nuevo en [DC03] se da un paso más hacia la concepción del marco de referencia unificado. Al igual que en el caso anterior, se presentan contribuciones para las dos fases: diseño y ejecución. Todas las aportaciones aparecen reflejadas en la figura 3.8 (2 pág. adelante).

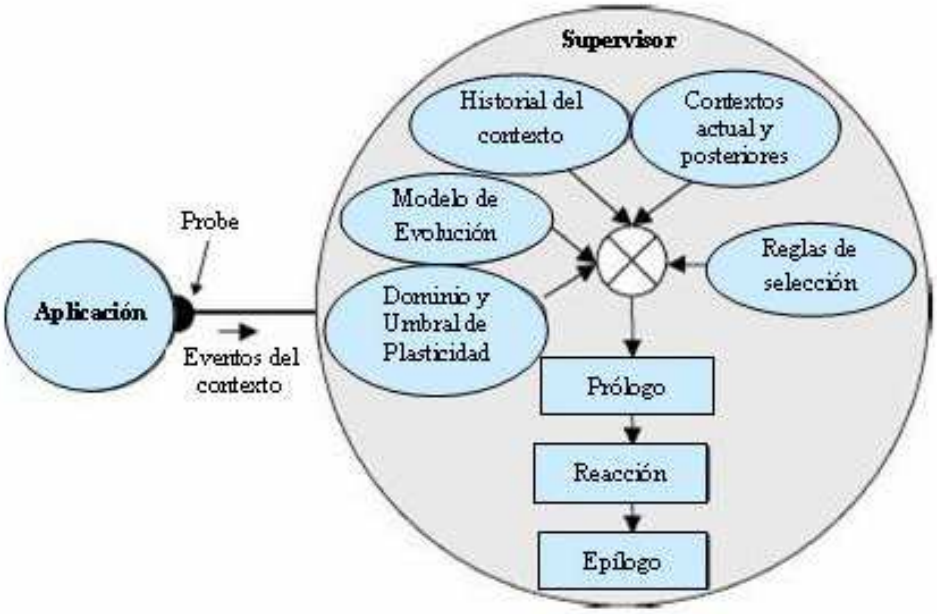


Figura 3.7: Proceso y mecanismo de adaptación en tiempo de ejecución utilizado en *Probe*.

Fase de diseño

Las aportaciones para la fase de diseño son las siguientes:

- Proponen un modelado concreto para especificar la evolución a través del *Modelo de Evolución*.
- Se incorpora un atributo más en la caracterización del contexto de uso: el usuario. En consecuencia, se incorpora un *Modelo de Usuario* en el conjunto de *modelos iniciales* (modelos especificados por el diseñador).
- Se añade el *Modelo de Transición* en el conjunto de *modelos iniciales*, basado en el trabajo de Barralon [Bar02]. Se trata de especificar la manera de procesar un cambio contextual.
- Por lo que respecta a los *modelos transitorios*, en el primer nivel de abstracción se substituye la *Especificación orientada a tareas* por una especificación de alto nivel de abstracción de las tareas y los conceptos, y que se conocerá posteriormente como el *Modelo de Tareas* y el *Modelo del Dominio* respectivamente.

Fase de ejecución

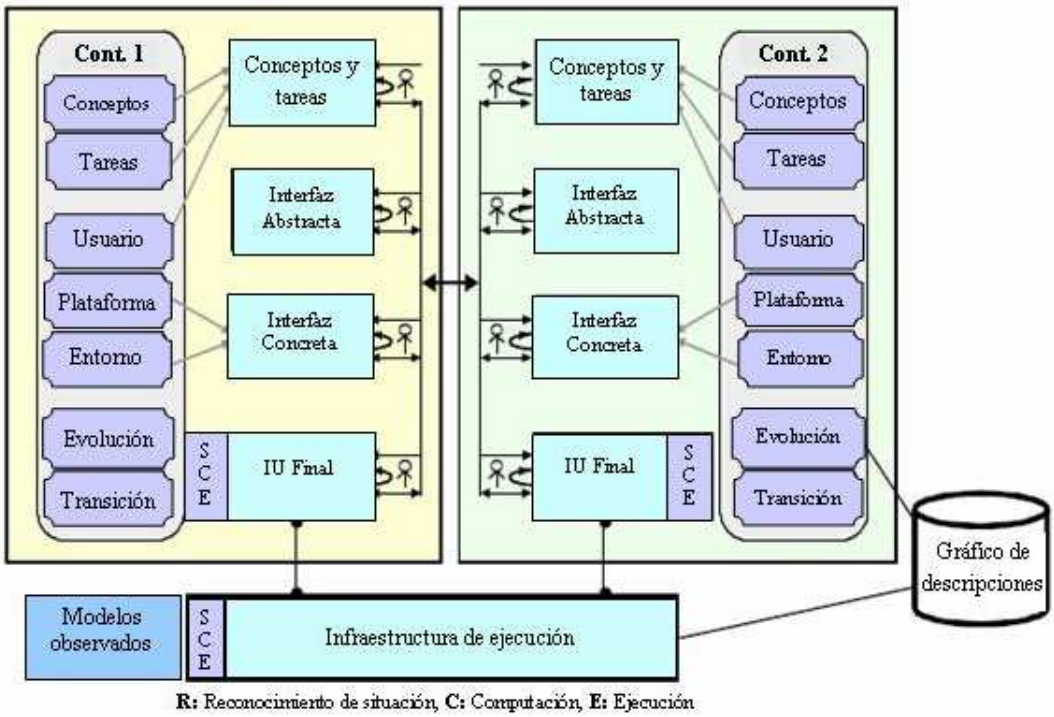


Figura 3.8: Marco de referencia resultante del segundo refinamiento.

Tal y como ya se introdujo en [CCT00] y ya ha sido comentado anteriormente, el mecanismo de software que soporta cambios contextuales en tiempo de ejecución se estructura en un proceso en tres pasos: (1) *reconocimiento de la situación*; (2) *computación de una reacción*; y (3) *ejecución de la reacción*. En [DC03] se propone un enfoque para resolver los pasos (2) y (3) consistente en el uso de un grafo de descripciones y de un conjunto de mecanismos para capitalizar las IUs a distintos niveles de abstracción, de acuerdo a la especificación de la adaptación en el caso de producirse un cambio en el contexto de uso. Esta especificación de la adaptación se encuentra representada en el *Modelo de Evolución*. Además, en esta contribución se propone un modelado para éste soportado en la noción de *grafo conceptual*[39].

3.3.5. El CAMELEON Reference Framework

Fruto del proyecto CAMELEON, el CAMELEON Reference Framework representa un marco conceptual de referencia que proporciona principios genéricos y unificados para

[39]Sistema de representación gráfica del conocimiento basado en nodos interconectados por arcos que toma la forma de un grafo, en concreto un grafo bipartido etiquetado.

estructurar y comprender el proceso de desarrollo de IUs multi-contextuales. Se puede considerar, por tanto, un marco de referencia unificado completo en el que se detallan los métodos y modelos que intervienen en cada etapa y de qué manera [CCT+02a].

3.3.5.1. Descripción general

Como se muestra en la figura 3.9, este marco de referencia se estructura en tres módulos: los modelos genéricos, el soporte para la fase de diseño y los mecanismos que actúan en la fase de ejecución. Por lo que respecta a los modelos, intervienen tres tipos distintos:

- los *modelos ontológicos*. Son modelos abstractos de los conceptos involucrados en el desarrollo de IUs multi-contextuales, los cuales pueden ser instanciados en *modelos arquetípicos* y/o *modelos observados*. Se trata por tanto de meta-modelos independientes de cualquier dominio y sistema interactivo. Se distinguen tres tipos de *modelos ontológicos*: los *del dominio*, *del contexto de uso* y *de adaptación*.
  - Los *modelos ontológicos del dominio* soportan la descripción de los conceptos y tareas de usuario relativos al dominio. Estos modelos mejoran los modelos del dominio tradicionales con el fin de acomodar las variaciones en el contexto de uso. Típicamente, los modelos de conceptos consisten en diagramas de clase UML enriquecidos con variaciones del dominio de conceptos [The01]. Un ejemplo de instanciación en *modelo arquetípico* sería el hecho de que para modelar el concepto mes puede recurrirse a un entero comprendido entre 1 y 12. Por otro lado, el modelo de tareas puede ser una descripción *ConcurTaskTree* [PMM97] enriquecida a través de decoraciones para especificar los contextos de uso para los que cada tarea toma sentido. Otro ejemplo sería expresar que la tarea 'escribir un artículo' carece de sentido en una PDA durante un trayecto en tren.
  - Los *modelos ontológicos del contexto de uso* caracterizan el contexto de uso en términos del usuario, la plataforma y el entorno, tal y como se propuso en [DC03]. De hecho, proporcionan herramientas para razonar acerca de estos atributos del contexto de uso, introducidos de forma conjunta para representar el contexto de uso físico.
    En particular, el *modelo de plataforma ontológico* proporciona herramientas para describir los recursos de computación que rodean la actividad. Por otro lado, se considera que *"el Modelo de Elementos de Interfaz propuesto inicialmente [TC99]* (véase *sección 3.3.1.1*)*, constituye uno de los aspectos del modelo de plataforma, con el fin de describir los elementos de interacción disponibles*

en una plataforma para representar la IU" [CCT+03].
El *modelo de entorno ontológico* identifica dimensiones genéricas para describir el entorno que rodea la actividad. El trabajo de Salber et al. [SDA98] es un intento en esta dirección.

- Los *modelos ontológicos de adaptación* proporcionan herramientas para describir la reacción cuando se produce un cambio en el contexto de uso. En concreto, el *Modelo de Evolución* ofrece un soporte para especificar la reacción en caso de un cambio de contexto de uso, y el *Modelo de Transición* soporta la manera de procesar esos cambios. Para ser más precisos, ofrece la oportunidad de especificar el *prólogo* y el *epílogo*, tratando de aliviar las discontinuidades ante cambios contextuales.

- los *modelos arquetípicos* son modelos declarativos que sirven como entrada para el diseño de un sistema interactivo particular, esto es, dependientes de un sistema interactivo y de un dominio determinado. Se trata por tanto de modelos *predictivos*, esto es, que modelan los contextos de uso previstos en la fase de diseño. Son especificados manualmente por el desarrollador.

- los *modelos observados* son modelos ejecutables que describen la realidad del momento, y por lo tanto guían el proceso de adaptación en tiempo de ejecución. Se trata por tanto de modelos *efectivos*, esto es, que modelan los contextos de uso que realmente se alcanzan en ejecución.

3.3.5.2. Proceso de desarrollo y ejecución

Al igual que en las versiones anteriores, el marco conceptual estructura el ciclo de vida en cuatro niveles de abstracción, estructurados con una relación de reificación que va del nivel más abstracto al nivel más concreto, así como una relación de abstracción que va del nivel concreto al nivel abstracto. De nuevo, con el fin de razonar acerca de la multi-contextualidad, el marco de referencia unificado considera dos conjuntos de *modelos arquetípicos*: el contexto 1 y 2, tal y como se muestra en la figura anterior.

Una vez instanciados los *modelos ontológicos* en *modelos arquetípicos* para un sistema interactivo y un dominio determinados, son utilizados como descripciones de entrada a tener en cuenta a lo largo de todo el proceso de desarrollo. En particular, cuanto más avanzado es el nivel de abstracción en el que se hace referencia a los *modelos del contexto de uso* (usuario, plataforma y entorno) más se amplía el dominio de multi-contextualidad.

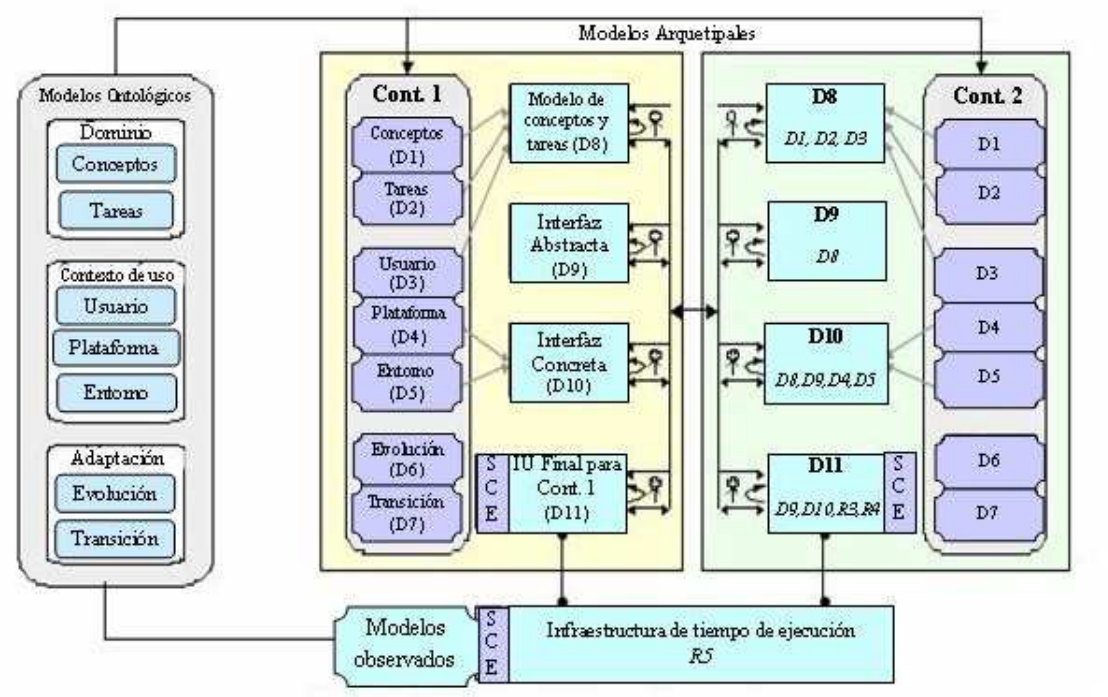


Figura 3.9: El Marco de Referencia Unificado propuesto en el proyecto CAMELEON.

Tal y como se puede observar en la figura 3.9, se utiliza una notación en cursiva para hacer explícito el ciclo de vida del conocimiento a lo largo del proceso de diseño. Así, en el nivel I, referente a un tipo de modelo, el modelo D_j se menciona si el conocimiento modelado en D_j está presente al nivel I, explicitando así las dependencias entre modelos y la trazabilidad del conocimiento. En particular, una I en el rango entre 1 y 7 identifica uno de los *modelos ontológicos*, y en el rango entre 8 y 11 un *modelo transitorio*. En concreto, los *modelos ontológicos* correspondientes al nivel del contexto de uso (nivel I en el rango entre 3 y 5) tienen dos posibles versiones: una versión de tiempo de ejecución (R_i), correspondiente a la situación observada durante el uso del sistema, y una versión correspondiente a la fase de diseño (D_i). Como se observa en la figura 3.9, los modelos de tiempo de ejecución sólo intervienen en el nivel 11 y en la fase de ejecución, puesto que hasta entonces se desconoce el contexto de uso realmente producido. Así, éstos influyen en la producción de la *IU Final* (nivel 11) y en todos los pasos que constituyen el proceso de ejecución: *reconocimiento de la Situación* (S), *Computación de la reacción* (C) y *Ejecución de la reacción* (E).

Como en versiones anteriores, el proceso utiliza una combinación de transformaciones verticales y horizontales para transformar los *modelos iniciales* en cada uno de los *modelos transitorios*, hasta derivar la *IU Final*, esto es, la IU expresada en código fuente, preparada

para ser interpretada o compilada. Igualmente, la transformación vertical puede consistir en un proceso top-down (*reificación*) o bottom-up (*abstracción*). Un ejemplo de transformación horizontal sería traducir descripciones de contenidos entre los lenguajes HTML y WML. Además, tal y como ya se introdujo en [CCT+02b], haciendo uso del operador de cruce es posible llevar a cabo una transformación horizontal y vertical al mismo tiempo, a modo de atajo que evita tener que producir modelos intermedios.

Con respecto a la fase de ejecución, el marco expresa cuándo, dónde y cómo un cambio de contexto es considerado y soportado en la IU gracias a una relación de traducción [CCT+03].

3.3.6. Marco de Referencia de soporte a la Plasticidad

Tal y como se expone en el *Capítulo 2* (véase *Capítulo 2*; *sección 2.1.1*), en 2004 se realiza una profunda revisión del concepto de plasticidad identificando varias razones para un nuevo replanteamiento, las cuales están basadas en la experiencia de los grupos participantes en el proyecto CAMELEON. Entre otras cosas, se toma en consideración la hipótesis científica siguiente: "*la adaptación ante los cambios contextuales puede impactar no sólo la capa de presentación, sino también el núcleo funcional del sistema interactivo*" [CCD+04]. Basándose en esta premisa proponen enfocar la aplicación de la plasticidad a un nivel de granularidad mucho más fino: el *elemento de interfaz*, en lugar de hacerlo a nivel de sistema interactivo como hasta entonces.

Como consecuencia, el marco conceptual es ahora replanteado a nivel de *elemento de interfaz* proporcionando una nueva noción: el *Comet*[40] (del inglés COntext sensitive Multi-target widgETS), que puede entenderse como un molde para adaptar *elementos de interfaz*. La primera aparición del término es en [DCCD03], presentándolo como una nueva clase de *elemento de interfaz* que soporta adaptación a cualquer nivel de abstracción: conceptos y tareas, IUs abstracta, concreta y final. En efecto, éste puede auto-adaptarse a una nueva situación contextual, ser adaptado por otra componente, así como ser dinámicamente descartado cuando deja de ser válido para cubrir el nuevo contexto de uso. Para conseguirlo, los *Comets* publican la calidad en uso que garantizan para un conjunto de contextos de uso, las tareas de usuario y conceptos del dominio que son capaces de soportar, así como en qué medida soportan adaptación.

En otras palabras, haciendo uso de C*omets* la plasticidad es expresada directamente en los *elementos de interfaz*, en lugar de hacerlo a nivel de sistema interactivo, tratando de hacer explícita la equivalencia funcional de los *elementos de interfaz* y reduciendo así la

[40]Elementos de interfaz introspectivos que publican la calidad en uso que garantizan para un conjunto de contextos de uso.

granularidad de la plasticidad. La idea es construir herramientas donde los *elementos de interfaz* que soportan las mismas tareas y conceptos sean agregados en un único *Comet polimórfico*[41]. En particular, estos autores se encuentran trabajando en la implementación de una herramienta que responde a estos requisitos: la herramienta "Plasturgy studio".

Por otro lado, con el objetivo de ofrecer la posibilidad de especificar también los requisitos concernientes a la preservación de las propiedades funcionales (por ejemplo, la consecución de las tareas), el ámbito de la definición de plasticidad se extiende reemplazando el término *usabilidad* por el de *calidad en uso* (ISO 9126-4), refiriéndose éste a la definición ISO [ISO91], tal y como se expone en el *Capítulo 2* (véase *Capítulo 2*; *sección 2.1.1*). Como consecuencia de ello, los modelos de calidad de la ISO se hacen explícitos a partir de ahora, incorporándose en el marco conceptual como parte de los *modelos iniciales*. Con ello el marco conceptual se va enfocando hacia un objetivo de plasticidad, más que a un mero objetivo de multi-contextualidad.

En base a todas estas consideraciones el marco conceptual es objeto de una profunda revisión. El esquema resultante es el que se representa en la figura 3.10.

3.3.7. Marco de Referencia de soporte a la Distribución, Migración y Plasticidad dinámicas

Por primera vez se propone una arquitectura global para soportar la ejecución de IUs plásticas, en concreto para resolver la *migración*[42] entre plataformas y la *distribución*[43] (conceptos definidos en el *Capítulo 2* –véase *sección 2.1.5*) en tiempo de ejecución, en el trabajo de Balme et al. [BDB+04]. Hasta entonces este problema sólo se había planteado bajo un soporte estático, esto es, en fase de diseño. Se trata de CAMELEON-RT (donde RT representa Run-Time), un modelo de referencia que integra los componentes funcionales necesarias para soportar todas las manifestaciones de la Distribución de las IUs a través de clusters heterogéneos dinámicos, la Migración dinámica y la Plasticidad (*DMP-middleware*) tanto a baja como a gran escala. Además, proporciona heurísticas preeliminares para facilitar la instanciación del marco para el despliegue de infraestructuras de ejecución. De hecho, puede ser instanciado de distintas formas para implementar infraestructuras de tiempo de ejecución adecuadas para un subconjunto del espacio del problema planteado. De hecho, en esta misma contribución se presentan dos casos de estudio: CamNote -un visualizador de transparencias que se ejecuta sobre una plataforma

[41] Un *Comet* que incorpora múltiples versiones de al menos uno de sus componentes.
[42] Transferencia de todo o parte de la IU a diferentes recursos de interacción (véase *sección 2.1.5*).
[43] Una IU es distribuida si utiliza recursos de interacción que están distribuidos a través de un cluster.

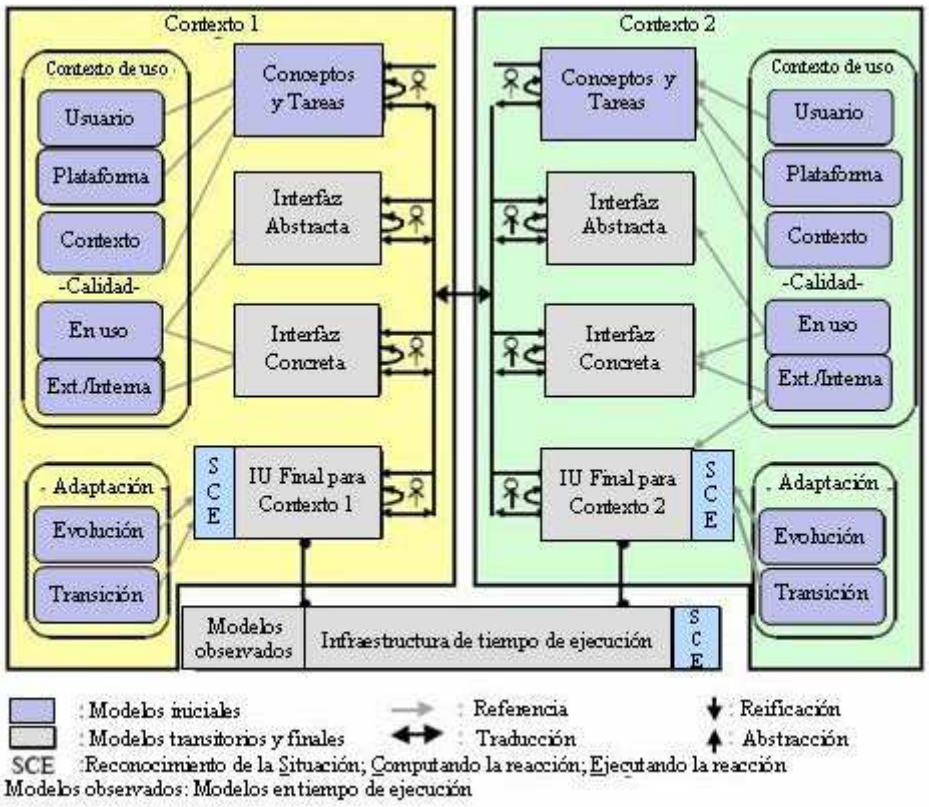


Figura 3.10: Marco de referencia para soportar IUs plásticas.

heterogénea dinámica- e I-AM (*Interaction Abstract Machine*) -un gestor de plataformas que soporta clusters de estaciones de trabajo heterogéneos dinámicos proporcionando una configuración dinámica de los recursos de interacción.

3.3.7.1. Estructura General

Como se muestra en la figura 3.11, CAMELEON-RT está estructurado en tres niveles de abstracción, que del inferior al superior son: (1) *capa de plataforma*; (2) *middleware* de distribución, migración y plasticidad, que proporciona todos los mecanismos y servicios para resolver las distintas situaciones planteadas; y (3) *capa de sistemas interactivos*.

La *capa de plataforma* (1) incluye no sólo información acerca del hardware y el sistema operativo, sino una amplia variedad de entidades físicas: superficies e instrumentos, facilidades de computación y comunicación, e incluso información sobre sensores y actuadores.

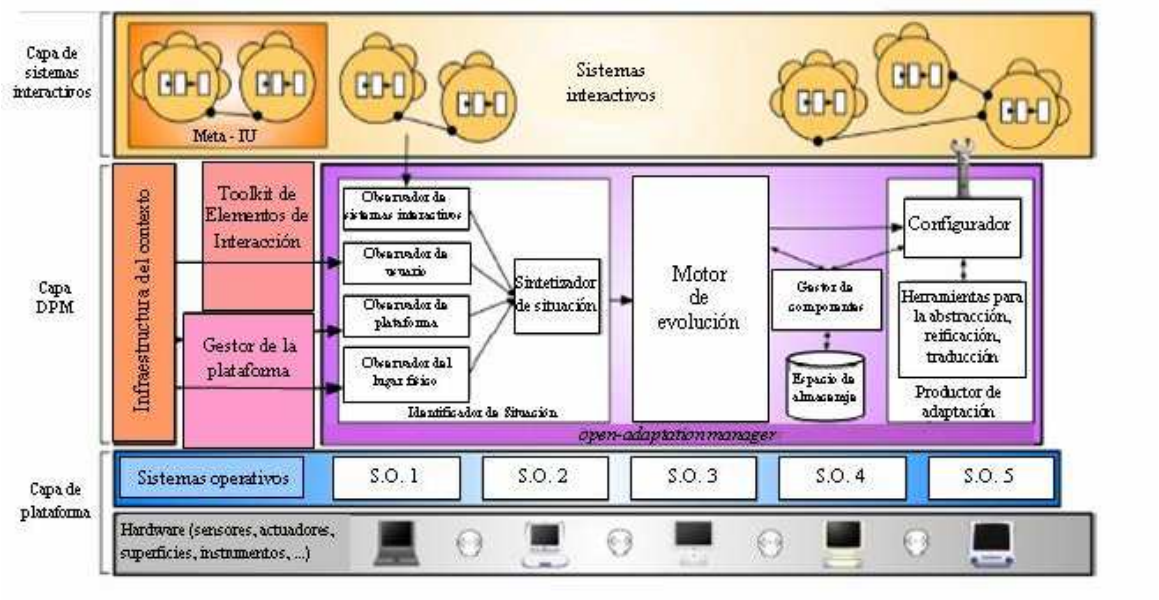


Figura 3.11: Modelo de referencia arquitectónico CAMELEON-RT.

La *capa de sistemas interactivos* (3) incluye los sistemas interactivos que los usuarios se encuentran ejecutando en el *espacio interactivo*[44].

La capa correspondiente al *middleware* de distribución, migración y plasticidad (2) satisface tres requisitos del espacio del problema: el modelado del espacio físico, el soporte para clusters heterogéneos dinámicos y la adaptación de la IU cuando se producen situaciones de distribución y migración. A cada uno de ellos le corresponde un servicio en esta capa que son, respectivamente: (1) una *infraestructura del contexto*; (2) un *gestor de plataforma*; y (3) un gestor de la interacción con los recursos externos que puedan surgir y que den lugar a la migración o a la distribución: el *open-adaptation manager*. La *infraestructura del contexto* permite al espacio interactivo construir y mantener un modelo del lugar físico. A partir de los datos provenientes de los sensores o de los eventos a bajo nivel del sistema operativo, es capaz de generar información contextual en el nivel apropiado de abstracción. Las herramientas *Context Toolkit* [DAS01] -descrita en el *Capítulo 5*- y *Contextors* [CR02] son ejemplos de herramientas válidas para implementar una infraestructura del contexto. El *gestor de plataforma* incluye un soporte para el descubrimiento de recursos, una abstracción para la heterogeneidad del sistema operativo y el hardware, y por último un soporte para la distribución y migración de IUs. En concreto,

[44]Combinación de tres componentes: (a) lugar físico donde la interacción toma lugar; (b) la computación, recursos de red y de interacción disponibles en ese lugar; y (c) el mundo digital o conjunto de servicios que soportan actividades humanas en ese espacio.

I-AM cubre estas tres funcionalidades. Utiliza la infraestructura *Contextors* para descubrir cambios en la plataforma.

Por último, el *open-adaptation manager*, que constituye una componente clave en CAMELEON-RT, incluye un conjunto de *observadores* especializados en distintos aspectos que alimentan un *sintetizador de la situación* con la información contextual necesaria. Éste informa al *motor de evolución* de la ocurrencia de una nueva situación que requiere un recurso externo (una *open-adaptation*). Se encarga entonces de elaborar una reacción utilizando un *gestor de componentes* y un *configurador* para producir una nueva IU. En particular, CamNote utiliza grafos conceptuales, tal y como se presenta en [DC03], para describir los componentes almacenados y expresar las peticiones para su recuperación. Según los autores, la literatura de sistemas auto-reconfigurables muestra que un enfoque por componentes reflexivos e introspectivos parece ser el más adecuado para el problema.

3.4. Comparativa de distintas herramientas existentes a través del Marco de Referencia Unificado

Como ya se ha mencionado, una de las motivaciones principales para definir un Marco de Referencia Unificado es disponer de un instrumento de ayuda a la comprensión del proceso de desarrollo de este tipo de IUs utilizado en cada caso, y con ello poder comparar distintos métodos, modelos y herramientas existentes. A continuación realizamos esta comparativa con un pequeño conjunto de herramientas como muestra significativa de las mismas, utilizando el CAMELEON Reference Framework presentado anteriormente. En particular, se ha elegido el trabajo desarrollado en el seno de dos grupos de investigación destacados en el campo de la plasticidad. El primero es el grupo ITSI[45] liderado por Paternò, que ha ido desarrollando sucesivas versiones de una herramienta inicial para IUs multi-plataforma, denominada TERESA. El segundo es el grupo de la Universidad de Hasselt (Expertise Centre for Digital Media[46]), liderado por Karin Coninx, que han venido desarrollando distintas técnicas para convertir descripciones de IU en XML a IUs concretas y dependientes del sistema -utilizando los lenguajes de descripción de IU en tiempo de ejecución que ellos mismos proponen en [LC01]-, teniendo en cuenta las restricciones que presentan los sistemas embebidos y los dispositivos móviles. La aplicación práctica de estas técnicas se materializa en las herramientas Dygimes, y posteriormente la arquitectura de ejecución para IUs sensibles al contexto DynaMo-AID, que se presentan a continuación.

[45]http://www.isti.cnr.it/ Instituto di Sciencia e Tecnologie dell'Informazione "A. Faedo". Instituto CNR (Consiglio Nazionale delle Ricerche)

[46]http://edm.uhasselt.be/about/the_institute

3.4.1. TERESA

TERESA (Transformation Environment for inteRactivE Systems representAtions) [PS03], [MPS03], [MPS04] es una herramienta orientada a Web diseñada para la generación de IUs para distintas plataformas en fase de diseño. En concreto genera XHTML para ordenadores de sobremesa, teléfonos móviles y también IUs VoiceXML. El método propuesto en TERESA parte de la especificación del *modelo de tareas* creado utilizando la técnica ConcurTaskTrees [PMM97], [MPS02] (en adelante CTT), cuya notación fue extendida para permitir a los diseñadores indicar en la especificación de cada tarea la plataforma –o conjunto de dispositivos con características similares- apropiada para soportarla, a través de un atributo de plataforma. CTT es una notación basada en el lenguaje formal LOTOS [ISO88] para el análisis y generación de IUs a partir de *modelos de tareas*, los cuales presentan una estructura jerárquica de las mismas. Para facilitar su especificación, CTT presenta una notación gráfica, que puede ser editada utilizando el editor gráfico CTTE[47] (ConcurTaskTrees Environment). Esta notación está siendo ampliamente utilizada en distintas metodologías como método para la especificación del modelo de tareas, y va camino de convertirse en el estándar de facto.

Una vez especificado el *modelo de tareas*, se genera la *IU Abstracta* en términos de su estructura estática (*modelo de presentación*) y comportamiento dinámico (*modelo de diálogo*), por medio de un análisis de las relaciones entre tareas. En concreto, la *IU Abstracta* se describe en términos de *objetos abstractos de interacción* [VB93], los cuales se transforman en *objetos concretos de interacción* una vez se selecciona la plataforma objetivo. La figura 3.12 representa gráficamente la instanciación del CAMELEON Reference Framework para el método propuesto en TERESA y su herramienta subyacente. Tal y como se observa, la traducción entre contextos de uso (concretamente la traducción a otra plataforma) se realiza exclusivamente en el nivel más alto de abstracción: el de tareas y conceptos. Esto proporciona una gran flexibilidad para soportar múltiples variaciones en las tareas de acuerdo a las restricciones impuestas por el contexto de uso. De hecho, los diseñadores deben filtrar el *modelo de tareas* y, si es necesario, refinarlo, de acuerdo a la plataforma objetivo, obteniendo un *modelo de tareas* específico para la plataforma.

En la figura se pueden observar también otro tipo de detalles. Por ejemplo, que la herramienta admite la intervención del diseñador en los niveles superiores de abstracción, con el objetivo de refinar las especificaciones para las distintas plataformas consideradas. Proporciona un solo punto de entrada en el nivel superior. Eso significa que el proceso únicamente puede partir del *modelo de tareas y conceptos*. Sigue una estrategia totalmente top-down donde cada nivel de abstracción considerado es descrito a través de un lenguaje

[47] http://giove.cnuce.cnr.it/ctte.html

basado en XML específico para cada nivel, inclusive para la especificación del modelo de tareas CTT. TERESA no proporciona ningún tipo de consideración para la fase de ejecución, de manera que todos los modelos contemplados intervienen en la fase de diseño. Por lo tanto, la *migración* entre plataformas que soporta es una migración estática, esto es, entre sesiones.

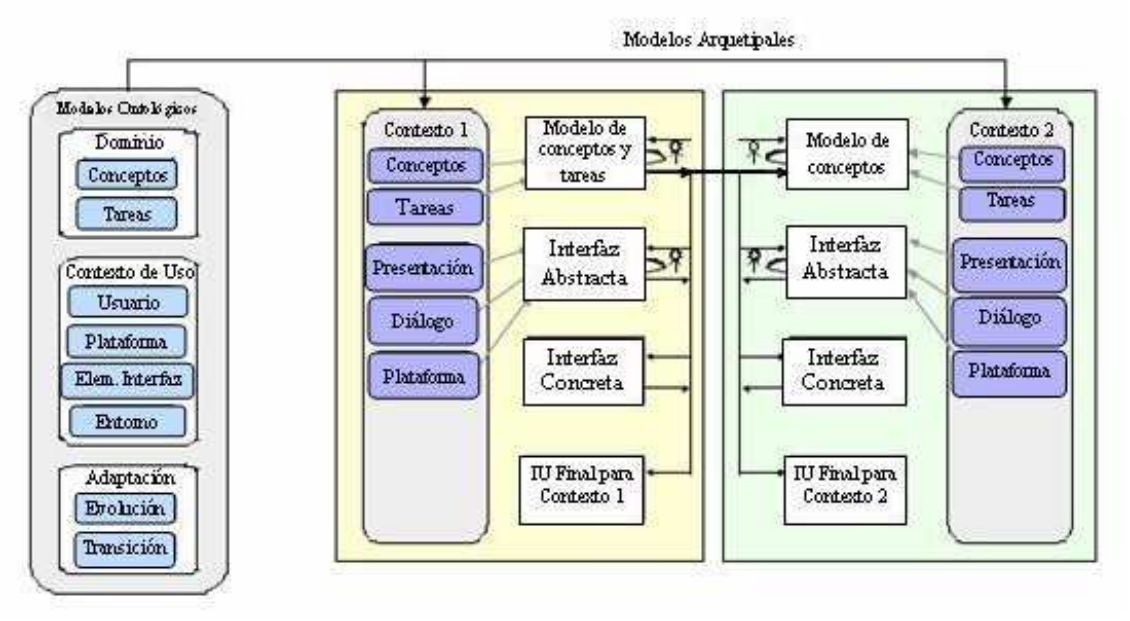


Figura 3.12: Marco de Referencia Unificado instanciado en TERESA.

TERESA explota el lenguaje de descripción de IUs TeresaXML, el cual soporta varios tipos de transformaciones entre modelos, siempre de un nivel más abstracto a un nivel más concreto.

En 2004 se implementa un nueva versión de la herramienta TERESA [CMP04], que flexibiliza la versión inicial de varias formas, tal y como se enumera a continuación: (1) introduciendo varios tipos de modificaciones en la *IU Abstracta*, algunos de los cuales comportan cambios radicales en las mismas; (2) añadiendo puntos de entrada tanto en el nivel abstracto como en el concreto (eso permite ignorar el nivel superior de abstracción); y (3) muy relacionada con la anterior, la habilidad para transformar el diseño para distintas plataformas (en particular, entre ordenadores de sobremesa y móviles) en niveles intermedios de abstracción, a través de un módulo de rediseño; y (4) el rediseño de una IU ya existente para adecuarlo a otra plataforma, partiendo del nivel concreto. En particular, este módulo de rediseño genera una descripción abstracta y concreta para dispositivos móviles del cual obtener la *IU Final* automáticamente, partiendo de la *IU Concreta* y de cierta información de la *IU Abstracta* de partida. Todas estas variaciones permiten soportar un

ciclo de desarrollo mucho más flexible. Además, en esta versión es posible introducir las IUs resultantes de aplicar ingeniería inversa a través de la herramienta Vaquita [BVS02], para la generación de nuevos diseños. La figura 3.13 (pág. siguiente) muestra una representación gráfica de esta versión de TERESA a través del marco de referencia unificado.

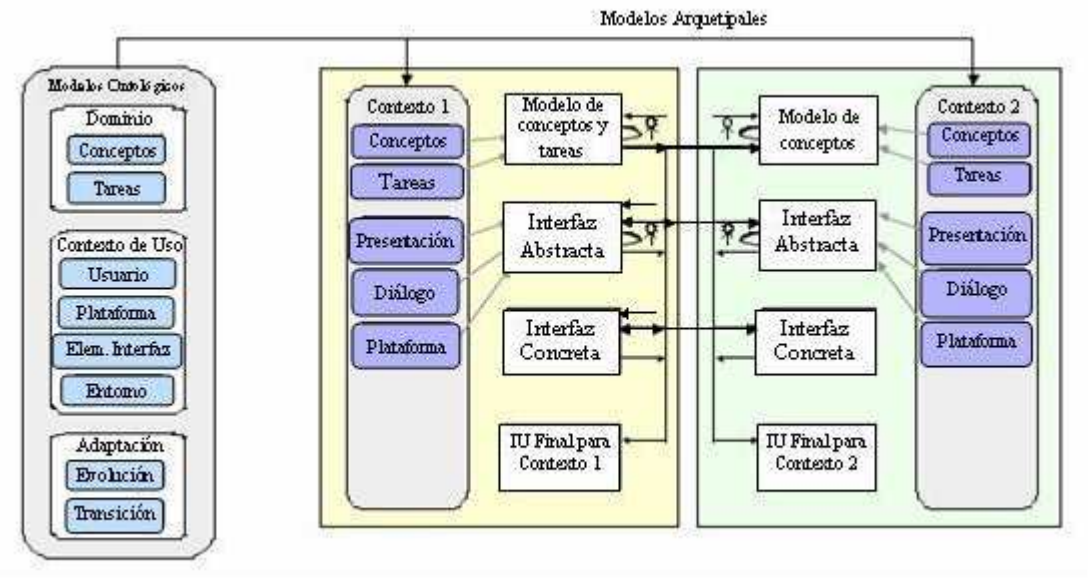


Figura 3.13: Marco de Referencia Unificado instanciado en una versión posterior de TERESA

Esta herramienta está públicamente disponible y puede descargarse gratuitamente[48].

3.4.2. Dygimes

Dygimes [CLV+03], [Luy04] es una metodología y una herramienta para la generación de IUs que cubre todas las fases del ciclo de vida. Además, soporta la generación en tiempo de ejecución de IUs para distintas plataformas móviles y sistemas embebidos, posiblemente equipados con distintos dispositivos de entrada/salida y, por lo tanto, ofreciendo múltiples modalidades. El generador de código que ofrece es capaz de transformar dinámicamente una especificación de tareas en una IU operativa, haciendo uso de algoritmos que se incluyen en la propia herramienta. Precisamente, fue concebida para facilitar el trabajo al diseñador y codificador de IUs para este tipo de sistemas, evitándoles el esfuerzo de tener que disponer de un conocimiento completo de cada uno de los dispositivos destino involucrados, lo que constituye el elemento distintivo de esta herramienta. Para conseguir esta

[48] http://giove.cnuce.cnr.it/teresa.html

flexibilidad y facilitar la reutilización de diseños ya existentes, esto es, hacer que el proceso de creación de IUs sea menos dependiente de las propiedades del dispositivo, utilizan descripciones de la IU basadas en XML.

El *modelo de tareas* utilizado sigue la notación CTT, que se guarda en formato XML para posteriormente ser anotado con información extra para poder ser operativo en el sistema desarrollado. En este sentido, se puede considerar que la especificación CTT es enriquecida con descripciones de IU basadas en XML. De nuevo, el entorno proporciona una herramienta de anotaciones que permite unir estas descripciones con las hojas del árbol CTT correspondiente, estableciendo así las conexiones necesarias entre ambas especificaciones. Este proceso aparece reflejado en la figura 3.14. En particular, la finalidad del manejador de restricciones de disposición espacial es la de asistir en el proceso de transformación de IUs para ordenadores convencionales en IUs para dispositivos compactos, un proceso en el que la mayoría de las técnicas existentes fallan al no obtener IUs mínimamente usables. Esta componente garantiza, al menos, que la IU es adecuada para el dispositivo y consistente con otras versiones. Las restricciones espaciales lineales que utiliza son también descritas en una sintaxis simple basada en XML.

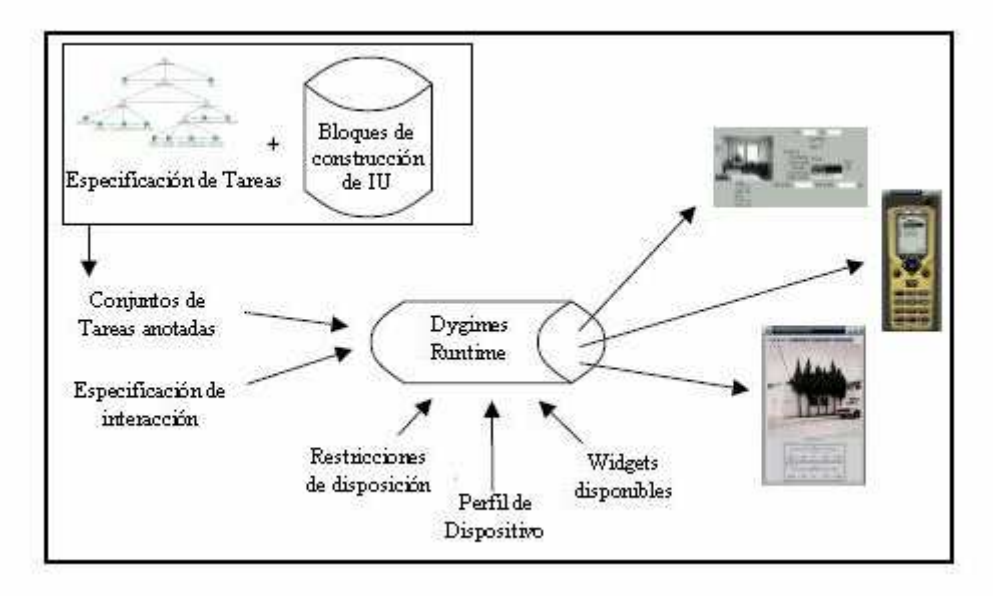


Figura 3.14: Proceso de creación de IUs multi-dispositivo en Dygimes.

Otros detalles de funcionamiento son que la especificación de tareas anotada contiene especificaciones para los distintos mecanismos de renderización (aspectos de presentación), y su comportamiento relacionado (aspectos de diálogo). Estas especificaciones se encuentran también escritas en un lenguaje basado en XML. En el paso siguiente se derivan

especificaciones dependientes de la plataforma a través de transformaciones XSLT. El resultado se conecta a una descripción de los dispositivos de entrada/salida de alto nivel de abstracción.

Se ha intentado representar el proceso de desarrollo de IUs seguido en Dygimes a través del marco conceptual unificado del que se dispone como referencia común, facilitando también de este modo la comparativa que se está realizando. El resultado es el que se recoge en la figura 3.15. Como puede observarse, la traducción entre plataformas se lleva a cabo en el nivel abstracto. Por lo tanto, el punto de entrada del proceso de reificación está localizado en dicho nivel (especificación de tareas). El proceso de renderización hasta obtener la IU final válida para una plataforma destino está totalmente delegado al generador de código dinámico, proporcionando la reutilización y la facilidad de migración entre plataformas deseada. No obstante, el diseñador podría optar por elegir y aplicar un conjunto de reglas de mapeo y de restricciones de disposición espacial más adecuado con el fin de obtener un mejor resultado para un dispositivo concreto (representado en la entrada "*plataforma*" y "*dispositivo*"). El hecho de ser únicamente una opción por parte del diseñador queda plasmado en la figura con un trazado más tenue y líneas discontinuas para las flechas (parte derecha de la figura 3.15).

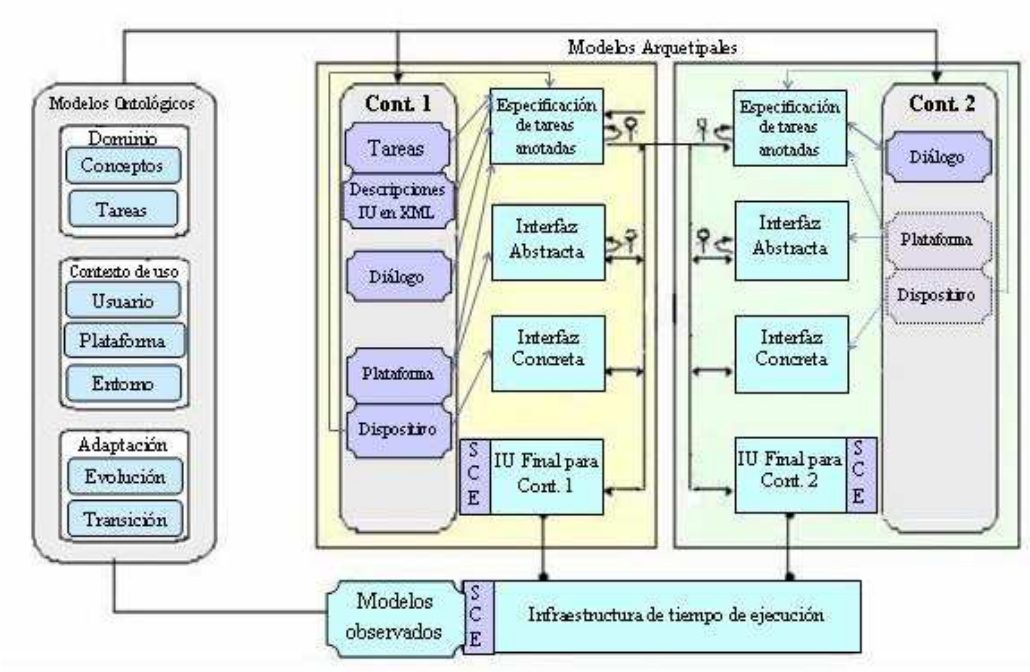


Figura 3.15: El marco de referencia unificado instanciado en Dygimes.

Otra novedad es que el modelo de diálogo requerido para generar la IU para la nueva plataforma es también generado automáticamente, indicando en qué orden aparecen los diálogos para el usuario, así como una descripción de las transiciones entre tareas. De nuevo, este paso se obtiene utilizando un algoritmo también incorporado en la herramienta. Este es el motivo por el que en la figura aparece una flecha bidireccional entre la especificación de la tarea y el modelo de diálogo en el contexto 2.

La herramienta Dygimes estuvo ya operativa y fue utilizada con éxito durante la realización del proyecto SEESCOA[49]. Cabe decir que utilizan un lenguaje de IU independiente de dispositivo desarrollado en el propio proyecto, denominado SeescoaXML que, además, soporta la migración en tiempo de ejecución de la IU [LLCR03].

3.4.3. DynaMo-AID

DynaMo-AID (**Dyna**mic **Mo**del-b**A**sed user **I**nterface **D**evelopment) [CLC04a] es un trabajo posterior a Dygimes que proporciona un soporte de diseño y una arquitectura de ejecución para IUs sensibles al contexto. En realidad, DynaMo-AID constituye una parte de la herramienta Dygimes. Las aportaciones de Dynamo-AID respecto a Dygimes son diversas: (1) la adaptación no sólo a la plataforma, sino también a las condiciones de red y del entorno; (2) la combinación de la información contextual en los distintos niveles del desarrollo de IUs basadas en modelos, extendiendo los modelos tradicionales para poder incorporar una componente del contexto y de este modo soportar el diseño de este tipo de IUs; y (3) la consideración del diseño de una IU para un servicio cuando éste pasa a estado disponible para la aplicación. Precisamente éste último punto es la principal contribución de la herramienta.

En particular, los distintos niveles de introducción de la información contextual son los siguientes: (a) en el *modelo de tareas* convirtiéndolo en un modelo dependiente del contexto, tal y como se propone en [MPS03], [PLV01], [SLV02], ampliando el ámbito de aplicación a otras clases de contexto, no sólo a la plataforma como en [MPS03]; (b) en el aspecto de navegación del *modelo de diálogo*, tal y como estos mismos autores proponen en [CLC04b], y también se analiza en [LVS00] y [VLF03]; y finalmente (c) en el nivel de presentación, con el fin de guiar la elección de los *elementos de interfaz* más apropiados, tal y como se propone en [VB93], [LC01]. En [CLC04b] se presenta la manera en que estos modelos son extendidos. La idea fundamental consiste en definir versiones dinámicas de cada tipo de modelo (*modelos dinámicos*), los cuales pueden cambiar en tiempo de ejecución, de manera que pueden incluso fusionarse con otros modelos del mismo tipo.

[49]Software Engineering for Embedded Systems using a Component-Oriented Approach. Periodo: 10/99-10/03. http://www.cs.kuleuven.ac.be/cwis/research/distrinet/projects/SEESCOA/

Así, el *modelo de tareas dinámico* en DynaMo-AID es un ConcurTaskTree al que se le ha añadido información relativa a la plataforma y al contexto.

A grandes trazos, el proceso a seguir consiste en la siguiente sucesión de pasos:

1. Enriquecer el *modelo de tareas* para obtener el *modelo de tareas dinámico* con la notación CTT.

2. Igual que en Dygimes, unir las descripciones de alto nivel acerca de los componentes de IU independientes de la plataforma a las hojas del árbol CTT, con el fin de establecer las conexiones necesarias entre ambas especificaciones, tal y como estos mismos autores proponen en [LCCV03], [CLC04b]. De este modo todas las posibles IUs disponen de una completa anotación de la especificación de tareas.

3. Calcular el ConcurTaskTrees Forest, que es la colección de los ConcurTaskTrees que describen las tareas a ejecutarse para cada contexto de uso.

4. Obtener el *modelo de diálogo dinámico* utilizando una iniciativa mixta; en primer lugar los *modelos de diálogo* se extraen automáticamente de cada ConcurTaskTrees correspondiente a cada posible contexto de uso, tal y como se propone en [LCCV03], y a continuación el diseñador establece las transiciones entre *modelos de diálogo* correspondientes a los cambios de contexto. De este modo se construye un *modelo de diálogo dinámico* que describe las transiciones entre estados de la IU tanto intra-contextuales como inter-contextuales, abarcando todas las transiciones que pueden darse durante la ejecución de la aplicación. Este aspecto constituye una de las aportaciones de la herramienta.

5. Añadir información del contexto uniendo el *Modelo de Entorno Dinámico*[50] al *Modelo de Tareas Dinámico* y al *Modelo de Diálogo Dinámico*. El *Modelo de Entorno Dinámico* representa los cambios contextuales proporcionando un modelo que describe la reacción a tomar ante estos cambios. Se trata, por tanto, de un modelo análogo al *Modelo de Evolución*, tal y como se introduce en [CCT00].

6. Obtener el *modelo de presentación dinámico* -la IU sensible al contexto- a través del proceso de transformación de los *objetos de contexto abstractos*[51] en *objetos de contexto concretos*[52].

[50] Entendido como un conjunto de *objetos de contexto abstractos* -un concepto análogo al de *objeto de interacción abstracto* de [VB93] para describir componentes de la IU independientes del contexto.

[51] Un objeto que puede ser interrogado acerca del contexto que representa.

[52] Objetos que encapsulan las entidades que representan una clase de contexto a través de las oportunas reglas de mapeo.

Además, como se ha indicado anteriormente, esta herramienta considera la aparición y desaparición dinámica de servicios, ofreciendo por tanto la posibilidad de agregar nueva funcionalidad a ser integrada con el resto de la aplicación. Así, cuando un servicio se hace disponible, tanto el *modelo de diálogo* como el *modelo del entorno* deben ser actualizados. En particular, el *modelo de diálogo* se extiende de manera similar a como se añade un nuevo contexto de uso, con el correspondiente enlace al correspondiente *modelo de tareas*. Por lo que respecta al *modelo del entorno*, se introducen nuevos *objetos de contexto abstractos*, los cuales deben ser mapeados en *objetos de contexto concretos*.

La figura 3.16 recoge de manera simplificada este proceso.

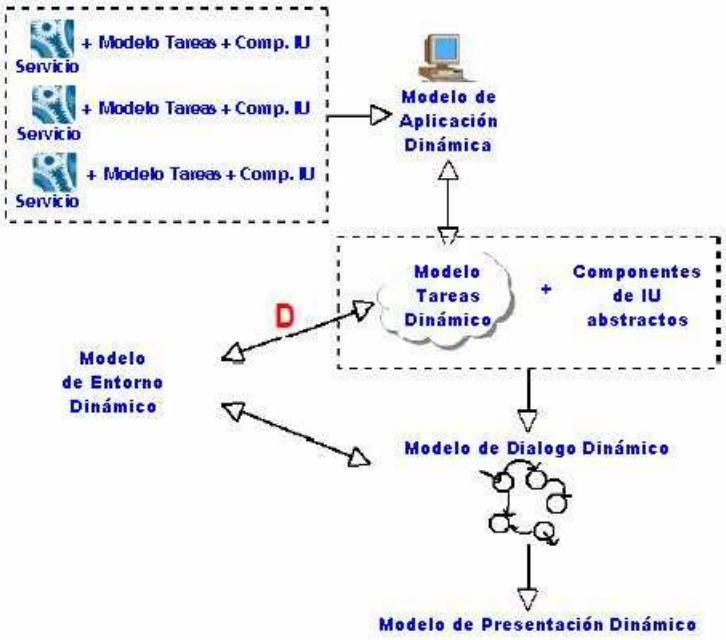


Figura 3.16: El proceso de diseño en DynaMo-AID.

Con el fin de ser contrastado con otras herramientas, como por ejemplo, para identificar las nuevas aportaciones respecto a Dygimes, ha sido plasmado el proceso descrito a través del modelo de referencia unificado. Es lo que se representa en la figura 3.17 (pág. siguiente).

3.5. Otras aproximaciones Basadas en Modelos para el desarrollo de IUs Multi-Contextuales

En el apartado anterior se ha recogido una representación de las herramientas existentes para el diseño de IUs basadas en modelos y se han contrastado entre sí haciendo uso del

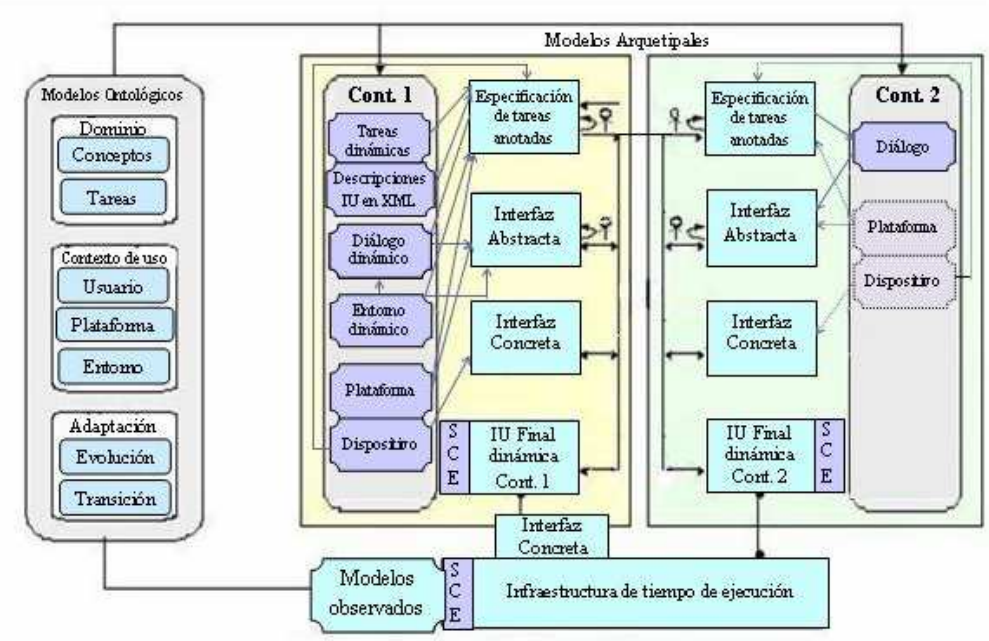


Figura 3.17: El marco de referencia unificado instanciado en DynaMo-AID.

marco de referencia unificado. A continuación se relacionan otras aproximaciones también basadas en modelos, convenientemente clasificadas de acuerdo al ámbito de aplicación o el propósito perseguido en cada caso.

3.5.1. Herramientas iniciales más influyentes

No podemos concluir este estudio de las técnicas basadas en modelos sin hacer hincapié en **TRIDENT** (Tools foR an Interactive Development ENvironmenT) [BHL+95], mencionada anteriormente, que constituye una herramienta de referencia dentro del enfoque basado en modelos. Propone la generación de IUs para distintas plataformas de la manera más automatizada posible. TRIDENT utiliza un modelo entidad/relación enriquecido, así como grafos de encadenamiento de actividades (ACG – Activity Chaining Graph) para ligar las tareas interactivas del usuario con la funcionalidad del sistema. Como ya se ha comentado anteriormente, lo más destacable de TRIDENT es la puesta en práctica de los conceptos o*bjetos abstractos de interacción*, o*bjetos concretos de interacción*, *ventanas lógicas* y *unidades de presentación* -conceptos introducidos anteriormente en este capítulo. Todo el proceso de diseño en TRIDENT es soportado por la herramienta SEGUIA (System Expert for Generating a User Interface Automatically)[53], que como su nombre indica, contiene un sistema experto que selecciona mediante reglas y heurísticas la mejor

[53]http://www.isys.ucl.ac.be/bchi/research/seguia.htm

traducción de OAIs a OCIs en el proceso de generación, además de soportar la disposición y el alineamiento de los OCIs en ventanas y cuadros de diálogo.

MOBI-D (MOdel-Based Interface Designer) [Pue98] ha sido otra de las propuestas basadas en modelos más influyentes en las propuestas actuales, como evolución de su antecesor, MECANO, proponiendo un ciclo completo dentro del desarrollo de IUs, y cubierto por un conjunto de herramientas. MOBI-D es un conjunto de herramientas para la generación de IUs desarrollado en la Universidad de Standford y financiado por DARPA (Defense Advanced Research Projects Agency)[54]. Las herramientas que componen MOBI-D son: (a) MOBI-D, el editor de especificaciones propiamente dicho. Permite establecer correspondencias (*mappings*) entre diferentes elementos de los diversos modelos; (b) UTel, una herramienta para el soporte a la elicitación de requisitos que será usada como paso previo a la fase de especificación; (c) TIMM, una herramienta de soporte a la decisión para la transformación de elementos abstractos en concretos; (d) MOBILE, herramienta para el diseño de la IU, el cual está soportado por el *modelo de tareas* y el *modelo de usuario*, construidos con las herramientas previas.

3.5.2. Mundo hipermedia

UWE (UML-based Web Engineering) [Koc00] es una metodología basada en modelos proveniente del ámbito hipermedial adaptivo que sólo utiliza modelos basados en UML para el modelado de la aplicación, completando la especificación del sistema con el uso del lenguaje OCL [WK99]. OCL es un lenguaje habitualmente utilizado en la especificación de restricciones dentro de los distintos diagramas UML. El aspecto que se enfatiza en UWE es el de la personalización. El proceso de desarrollo propuesto es soportado por la aplicación ArgoUWE [KKMZ03], herramienta que se apoya en ArgoUML para su implementación.

OO-H (Object Oriented Hypermedia Mehtod) [Cac03] es un método diseñado para la creación de aplicaciones Web. Se basa en tres modelos principales: el Diagrama de Acceso Navegacional, el Diagrama de Presentación Abstracta, que refleja cómo será la estructura y las relaciones de cada página Web y por último el Diagrama de Diseño Visual, que consiste en una presentación más concreta donde se define cómo se va a realizar la visualización de los distintos constructores utilizados. Por otra parte, utiliza diagramas de clases para modelar el dominio del problema. Todo el proceso de diseño puede ser realizado a través de la herramienta VisualWADE[55].

OOWS (Object Oriented Web Solutions) [PAF00] es un trabajo que explora la especificación de aspectos navegacionales en entornos Web como complemento a OO-Method

[54]http://www.darpa.mil/
[55]http://www.visualwade.com

[PIP+97]. Proporciona un método para el desarrollo automático de aplicaciones Web basado en el lenguaje OASIS [LRSP98].

Finalmente, dentro del ámbito de herramientas comerciales podemos encontrar Olivanova[56], basada en OO-Method [PIP+97]. Esta herramienta permite la generación de aplicaciones basadas en formularios automáticamente para distintos lenguajes destino, a partir de un detallado modelado conceptual. El código generado puede ser compilado para producir la aplicación final.

3.5.3. Migración dinámica

Como herramientas que, al igual que DynaMo-AID, dan solución a la migración entre plataformas en tiempo de ejecución, podemos nombrar Digestor [BS97], que produce diferentes IUs en tiempo de ejecución para visualizar páginas Web HTML en función de las restricciones impuestas por la plataforma. FlexClock [GRV01], que produce una apropiada *IU Concreta* en tiempo de ejecución, en función del tamaño de la pantalla cambiando la entrada numérica por un teclado virtual cuando el teclado físico no está disponible. El trabajo de Bandelloni en [BP04], es un servicio orientado a Web que soporta migración dinámica para las IUs desarrolladas por la herramienta TERESA, y que residen en la máquina servidor. Cuando una plataforma solicita migración, la plataforma origen envía una petición al servidor de migración, que explota tanto la información de contexto estático como dinámico para llevar a cabo el proceso de mapeo de la presentación. La página correspondiente y el contexto de ejecución para el dispositivo objetivo son enviados a la plataforma objetivo, que abrirá una ventana permitiendo al usuario continuar con su interacción. En este caso se puede decir que la inteligencia de la adaptación reside en el servidor, que es quien recoge los datos en tiempo de ejecución. Los datos dinámicos de la aplicación se utilizan para alcanzar *continuidad de la interacción*[57], mientras que la información de los diferentes tipos de plataforma involucrados es utilizada para adaptar la apariencia de la aplicación y su comportamiento en el dispositivo específico.

Otra iniciativa, que además de ser multi-plataforma es multi-modal, es la de [BP05]. También utiliza representaciones basadas en XML, y el *modelo de tareas* a través de la notación CTT. No obstante, la más similar a DynaMo-AID es iCrafter [PLF+01], puesto que, al igual que en aquélla, un generador de código construye dinámicamente *IUs Concretas* partiendo de una *IU Abstracta*.

[56] http://www.care-t.com

[57] Capacidad de migrar entre plataformas sin tener que reiniciar la aplicación en el nuevo dispositivo, esto es, continuando directamente la interacción en el nuevo dispositivo justo en el mismo punto donde se dejó.

Por lo que respecta a la distribución, existen diversas herramientas que soportan un cierto grado de distribución. No obstante, o bien la distribución de la IU obtenida es asignada estáticamente, o bien el cluster distribuido es estático y homogéneo, no respondiendo por tanto a un grado deseable de dinamismo. Un aspecto muy relacionado con la distribución, el cual es contemplado en el Modelo de referencia arquitectónico CAMELEON-RT, es el descubrimiento de recursos computacionales. Podemos citar el proyecto Aura (Carnegie Mellon University) [SG02] como infraestructura del contexto que cubre este aspecto. Sin embargo, está enfocada hacia plataformas elementales, y no clusters, y por lo tanto no aborda la distribución.

3.5.4. Migración estática

La práctica actual utiliza técnicas de *transcoding*, una técnica utilizada en el mundo Web para adaptar y convertir contenido Web -generalmente código HTML- a un formato apropiado para la creciente diversidad de dispositivos provistos de acceso a Internet. *WebSphere*[58] de IBM es un ejemplo significativo de editor de *transcoding*. Se trata de un software basado en servidor que traduce dinámicamente contenido Web y aplicaciones en múltiples lenguajes de marcado optimizando el resultado previo a la entrega a distintos tipos de dispositivos móviles. En su versión más simple, el significado semántico es inferido de la estructura de la página Web. Asimismo, la página Web es transformada utilizando esta información semántica. Una versión más sofisticada de esta técnica anota los elementos estructurales de la página Web, aplicando la transformación en base a estas anotaciones. Existen numerosos ejemplos de utilización de *transcoding*. Uno de ellos es la automatización de la traducción de HTML a WML. Aunque son técnicas de bajo coste, se basan normalmente en análisis sintáctico y transformaciones, produciendo resultados pobres en términos de usabilidad, puesto que tienden a proponer el mismo diseño en dispositivos con distintas posibilidades de interacción.

Florins y Vanderdonckt introducen en [FV04] la noción de *graceful degradation*, como un método para diseñar IUs usables para sistemas multi-plataforma cuando las capacidades de cada plataforma son muy diferentes. El enfoque se basa en un conjunto de reglas de transformación aplicadas a una única IU diseñada para la plataforma menos restringida. Uno de los mayores puntos de interés de este método es el de garantizar una máxima continuidad entre las versiones de las IUs específicas de cada plataforma. La motivación por el nombre es la siguiente. Como las reglas de transformación toman como punto de partida una IU personalizada para una pantalla extensa y producen IUs más reducidas, el proceso de transformación recibe el nombre de *degradación*. Por el hecho de que existe una

[58] http://www-306.ibm.com/software/pervasive/transcoding_publisher/

preocupación por producir IUs altamente usables adaptadas a las plataformas específicas mientras se preserva la consistencia entre versiones, a esta degradación se le denomina *degradación "agraciada"*. Este enfoque se basa en un conjunto de herramientas, descritas y clasificadas en un *framework* basado en modelos.

3.5.5. Ingeniería inversa

Existen muy pocas herramientas basadas en modelos que incluyan una operación de abstracción capaz de realizar ingeniería inversa. La herramienta pionera en este terreno es Vaquita[59] [BVS02], que reconstruye la descripción concreta de páginas Web en HTML a modelos de presentación. Como ya se ha comentado anteriormente, la herramienta TERESA es capaz de obtener nuevos diseños como resultado de enlazar el proceso resultante de la ingeniería inversa obtenida a través de Vaquita. ReversiXML[60] es la versión online de Vaquita. Es capaz de transforma una página Web en HTML a UsiXML, tanto al nivel de *IU Abstracta* como al de *IU Concreta*, así como de traducir una página Web de una plataforma a otra [BVC04]. Otro ejemplo es WebRevEnge[61] [PP02], que reconstruye el *modelo de tareas* correspondiente a un sitio Web en HTML.

3.6. Resumen y conclusiones del capítulo

A lo largo de la última década se han ido presentando distintas aproximaciones para el diseño de IUs basado en modelos, donde se soporta en mayor o menor medida la generación automática de la IU. En este capítulo se presenta en detalle este enfoque, contrastándolo con otras alternativas y trabajos relacionados en el campo de la especificación y automatización de la IU. Aunque en general estas herramientas han consistido en prototipos de las herramientas CASE para soportar los métodos de diseño y especificación propuestos, se empiezan a encontrar algunos ejemplos de herramientas comerciales basadas en modelos, como *WebRation*[62], *VisualWADE*[63] u *Olivanova* –anteriormente mencionada. Esto significa que la tecnología MB-UIDE empieza a alcanzar la madurez necesaria para ser comercializada como producto.

Numerosas técnicas, métodos y modelos sustentan este enfoque, y día a día se sigue innovando en la línea de integrar las consideraciones acerca del contexto de uso, en un intento por obtener herramientas altamente flexibles, adaptables y automáticas. En este

[59] http://www.isys.ucl.ac.be/bchi/research/vaquita.html
[60] http://giove.cnuce.cnr.it/webrevenge.html
[61] http://giove.cnuce.cnr.it/webrevenge.html
[62] http://www.webratio.com
[63] http://www.visualwade.com

capítulo hacemos buena cuenta de ello. A pesar del gran avance conseguido –entre ellos el desarrollo de un marco de referencia unificado-, no se ha dado solución, sin embargo, a numerosos problemas que siguen planteando este tipo de técnicas. Se ha hipotetizado que los sistemas basados en modelos no sean todavía suficientemente expresivos [PEGM94]. Se requiere, todavía, una mejora sustancial con el fin de incrementar la aceptación de los MB-UIDEs al nivel de otras herramientas de desarrollo de IUs especializadas. En la presente tesis se considera que el conjunto de modelos que se utilizan es limitado. En particular, la participación de un modelo contextual debería hacerse más extensiva. El manejo de los aspectos del contexto tanto en la fase de diseño como en la de ejecución puede contribuir a la anticipación a los cambios contextuales, aspecto que generalmente queda sin resolver.

En conclusión, aunque existen productos comerciales que utilizan esta clase de herramientas, aún existen aspectos que deben ser estudiados para incrementar su aceptación.

Capítulo 4

Marco Conceptual de desarrollo de IUs propuesto: el *Framework de Plasticidad Explícita Colaborativo*

"... ... users have different preferences with respect to font size, so ensure that your designs work well with both larger and smaller fonts than your personal preferences."

Jakob Nielsen. "Designing Web Usability" (pp. 29)

Este capítulo está destinado a describir el marco conceptual propuesto en la presente tesis como instrumento de referencia para la comparación, estudio y análisis de las distintas herramientas basadas en modelos, al que se le denomina *Framework de Plasticidad Explícita Colaborativo* (abreviadamente *FPE-C*). En la primera sección se describe en detalle presentando sus componentes, fases de desarrollo, operaciones y enfoques bajo los cuales ha sido concebido. En la sección siguiente se ofrece una comparativa con el *marco de referencia unificado* denominado CAMELEON Reference Framework, el cual ha sido presentado en detalle en el *Capítulo 3*. Se analizan las diferencias así como las contribuciones aportadas. En la sección siguiente se pone en práctica el uso del marco conceptual realizando el ejercicio para el que ha sido concebido: servir de soporte para el estudio y comparación entre herramientas. Se han escogido, además de las herramientas ya estudiadas en el *Capítulo 3*, otras que están estrechamente ligadas con la concepción del marco conceptual propuesto, y que han sido fuente continua de inspiración. Se trata de las herramientas Teallach y AB-UIDE. Además, este conjunto de herramientas ofrece una muestra suficientemente representativa y diversa en cuanto a las características y modelos de proceso seguidos en este tipo de entornos.

4.1. El Marco Conceptual propuesto: el *FPE Colaborativo*

El interés de este trabajo se centra en la definición de un marco conceptual de referencia para la construcción de herramientas *basadas en modelos* para el desarrollo de IUs plásticas que, además, tienen en cuenta las necesidades en curso relativas al trabajo en grupo, siguiendo el enfoque de la *visión dicotómica de plasticidad* propuesto en la presente tesis (véase *Capítulo 2*; *sección 2.2.*). La repercusión de este último aspecto, tal y como se detalla a continuación, implica que el marco conceptual se centra exclusivamente en la resolución de aquellas adaptaciones que conllevan una reconfiguración de la IU, y que por tanto requieren una adaptación y/o reexplotación de parte de su *modelado de interfaz*.

Tal y como se introduce en el *Capítulo 2*, este tipo de herramientas, que bajo la *visión dicotómica de plasticidad* actúan en el lado del servidor de una arquitectura cliente-servidor, se denominan en esta tesis *Motores de Plasticidad Explícita* (véase *sección 2.2.4.2.*), puesto que es el tipo de herramientas que dan soporte a la *plasticidad explícita*, según la *Definición 2.5* (véase *sección 2.2.2.*). Del mismo modo, al servidor que alberga el *Motor de Plasticidad Explícita* se le llama *servidor de plasticidad*. Finalmente, el marco conceptual de referencia propuesto recibe el nombre de *Framework de Plasticidad Explícita*, que al contemplar también situaciones de trabajo en grupo y el consecuente tratamiento de la información generada durante el transcurso de actividades colaborativas, se le añade el calificativo de *Colaborativo*. En efecto, este marco conceptual trata de representar no sólo las capacidades de adaptación, sino también de ofrecer una orientación para la integración de aspectos relativos al trabajo en grupo, con el fin de integrar también un soporte para la colaboración en este tipo de herramientas. Básicamente, se pretende contribuir a una adecuada y fluida interacción entre los miembros del grupo que conlleve a la consecución de una meta común, de acuerdo a una *consciencia de conocimiento compartido* (del término inglés *shared-knowledge awareness*).

El marco conceptual propuesto va acompañado de un conjunto de heurísticas, algunas de ellas recopilatorias de algunas ideas y observaciones empleadas recurrentemente, y que se ha considerado adecuado reunir aquí; otras noveles, propuestas por primera vez en la presente tesis.

Antes de entrar en detalle es conveniente introducir el concepto de *IU sensible al grupo*, para formalizar esa nueva necesidad mencionada anteriormente de incorporar las consideraciones relativas al trabajo en grupo en la generación de la IU y caracterizar aquellas IUs que la satisfagan.

Definición 4.1 (IU sensible al grupo): IU que (1) ha sido personalizada a las peculiaridades del entorno colaborativo para el que va destinada y al estado del trabajo en

grupo, aspectos que han sido tomados en consideración a lo largo del proceso de desarrollo; y que, adicionalmente, (2) puede ir provista de cierta capacidad para detectar situaciones que atañen al grupo y reaccionar ante las mismas, con el propósito de promover la colaboración también durante el proceso de interacción. □

En concreto, de esta segunda parte se encarga el *Motor de Plasticidad Implícita*, el cual se introduce en el siguiente capítulo.

Cabe señalar también que el marco conceptual, entendido como instrumento de referencia para el estudio de herramientas existentes, interesa que sea lo más versátil posible de cara a poder reflejar diversos modelos de proceso, y de ese modo resultar válido como elemento comparativo. En este sentido se contempla la posibilidad de seguir tanto un proceso de derivación puro –obtención de la IU a partir de un diseño a un alto nivel de abstracción, es decir, en *top-down*- como un proceso de abstracción puro –capaz de inferir diseños de la IU a un alto nivel de abstracción a partir de IUs finales, es decir, en *bottom-up*-, como cualquier combinación resultante de la intercalación de pasos de ambas modalidades –en este caso se hablaría de un proceso de construcción de la IU multi-direccional. Con objeto de diferenciar de la mejor manera posible cada paso y de identificar a lo largo de la explicación a qué proceso se está haciendo referencia se realizan las oportunas distinciones al respecto.

4.1.1. Principios sobre los que se sustenta

Para empezar la descripción del marco conceptual propuesto se introducen los fundamentos en los que se basa la definición del mismo.

4.1.1.1. Enfoque de Visión Dicotómica de Plasticidad

Tal y como se presenta en el *Capítulo 2* (véase *sección 2.2.*), la *visión dicotómica de plasticidad* promueve la separación tanto conceptual como operativa de los dos retos de plasticidad identificados en la presente tesis y formulados a través de los conceptos de *plasticidad explícita* y *plasticidad implícita* (véase *sección 2.2.2.*).

Siguiendo este enfoque, el *Motor de Plasticidad Explícita* tan sólo es responsable de resolver las adaptaciones en la IU que comportan cierta envergadura, al requerir una repetición total o parcial del proceso de generación de una IU acomodada a la nueva situación alcanzada tras un *cambio contextual* (*Definición 2.4*; *sección 2.1.3.*). Las adaptaciones ante *variaciones dinámicas en el contexto de uso* (*Definición 2.3*), que según la *Hipótesis 2 –sección 1.3-* es preferible resolver en tiempo de ejecución de manera proactiva, no

recaen sobre el *servidor de plasticidad*, sino que son responsabilidad del *Motor de Plasticidad Implícita*, ubicado en la plataforma cliente, como herramienta que da soporte a la *plasticidad implícita* (*Definición 2.7*; *sección 2.2.2.*).

Así, las únicas adaptaciones a resolver en el *servidor de plasticidad* son aquellas que de acuerdo a la *visión dicotómica de plasticidad* no son resolubles en la plataforma cliente, dada su envergadura. Estas adaptaciones son explícitamente requeridas mediante la emisión de una petición –de ahí el nombre de *'explícita'*. No obstante, con el propósito de complementarse con el *Motor de Plasticidad Implícita*, el *Motor de Plasticidad Explícita* debe estar provisto de un soporte adicional que le permita ponerse al día de los cambios sucedidos desde la última generación de una IU para el sistema. Sólo tras un proceso de actualización podrá dar respuesta a las nuevas necesidades de adaptación planteadas, a ser posible en tiempo de ejecución. Este soporte se vale de: (1) la capacidad intrínseca de un servidor de comunicarse con la/s plataforma/s cliente (recepción de peticiones y envío de respuestas –en este caso IUs- de nuevo a la plataforma cliente); y (2) un mecanismo de actualización de los modelos subyacentes a los cambios producidos. Para soportar la ejecución de este segundo aspecto en el *servidor de plasticidad* se aplica el *enfoque de modelos compartidos*, presentado a continuación. La unión de ambos mecanismos proporciona un medio para la propagación de cambios, considerado en esta tesis indispensable para alcanzar una *continuidad de plasticidad* [CCT01a] (véase *Capítulo 2*; *sección 2.1.5.*), especialmente en entornos basados en la *visión dicotómica*.

En este sentido, el marco conceptual propuesto, a diferencia del *marco de referencia unificado* que se tiene como referente (el CAMELEON Reference Framework), se focaliza exclusivamente en la explotación del *modelado de interfaz*. La parte correspondiente al procesamiento de las reacciones a llevar a cabo durante la ejecución y uso de la IU, también contemplada en el *marco de referencia unificado*, queda delegada en la infraestructura del cliente (a través del denominado *Motor de Plasticidad Implícita*).

Por último, cabe remarcar aquí que no siempre el proceso de reconfiguración de la IU a abordar en el *servidor de plasticidad* representará tener que interrumpir la ejecución en el lado del cliente. Esto es, el envío de una petición por parte del cliente, la reexplotación del *modelo de interfaz* y la recepción y puesta en marcha en el cliente de la nueva IU obtenida puede resolverse, en determinados casos, en una misma sesión –esto es, sin perder el contexto de la ejecución-, a pesar de que no se proporcione una respuesta en tiempo real. De hecho, esta situación representa el proceso de generación dinámica de IUs para resolver situaciones de *migración dinámica* (véase *Capítulo* 2; sección2,1,5.). En conclusión, la intervención del *Motor de Plasticidad Explícita* no debe concebirse como una operativa relegada a la fase de diseño, esto es, a resolver necesariamente entre sesiones.

4.1.1.2. Enfoque basado en modelos

Tal y como se justifica en el *Capítulo 3*, el enfoque por excelencia para desarrollar sistemáticamente IUs plásticas es el *enfoque basado en modelos*. La descripción del mismo, así como el proceso de diseño seguido en este tipo de herramientas se presenta en detalle en ese mismo capítulo. No obstante, se considera oportuno sintetizar aquí la esencia de los mismos recordando lo siguiente:

- Permiten al diseñador especificar las IUs a un alto nivel de abstracción, trabajando únicamente con descripciones lógicas, y dejando que los detalles de implementación sean proporcionados por el sistema, haciendo posible, además, una generación total o parcial de la IU de forma sencilla cuando los requisitos cambian.
- Las tres características principales que exhiben este tipo de entornos son las siguientes: (1) soporte de generación automática de IUs; (2) utilización de modelos declarativos para especificar la IU; y (3) la adopción de una metodología para soportar el desarrollo de la IU.

Es evidente que bajo estas premisas se reduce considerablemente el esfuerzo requerido por el desarrollador para obtener múltiples versiones de una IU para distintos dispositivos interactivos, modalidades de interacción o contextos de uso.

Uno de los pilares del proceso de diseño que caracteriza un *enfoque basado en modelos* es que se fomenta la *separación de conceptos*[1], aplicando un diseño progresivo de la IU a través de sucesivos niveles de abstracción. En particular, varios estudios [Sze96], [BDB+04] han coincidido en recomendar tres niveles conceptuales principales: el *modelo de tareas*, donde se consideran las actividades a llevar a cabo por los usuarios; el nivel abstracto, que trabaja con una descripción de la IU independiente de la modalidad y de la plataforma; y finalmente el nivel concreto, que proporciona una descripción lógica más refinada, y por lo tanto dependiente de la modalidad. No obstante, esta última descripción no consiste en una versión operativa, sino que es una descripción independiente de la plataforma. Todavía se requiere, por tanto, otro paso de concreción para seleccionar los componentes de interfaz finales, propios de la plataforma objetivo, los cuales van a formar parte del código ejecutable.

El marco conceptual propuesto se estructura también en cuatro niveles de abstracción, tal y como se propone en el proyecto Europeo CAMELEON [CCT+03]. Del nivel más

[1]Proceso de dividir un programa en distintas características o puntos de interés que se solapan en funcionalidad en unidades tan pequeñas como sea posible.

abstracto al nivel más concreto son: (1) *nivel superior de abstracción* en el que intervienen el *modelo de tareas*, los conceptos de dominio -formalizados a través del *modelo de dominio*- y por último el *modelo de diálogo*, como novedad propuesta en esta tesis en relación al CAMELEON Reference Framework; (2) el *nivel abstracto*, que maneja lo que se denomina a partir de ahora *IU Abstracta*; (3) el *nivel concreto*, que maneja esa descripción más refinada de la IU denominada *IU Concreta*; y (4) un último nivel de concreción, correspondiente al nivel físico, y al que se le denomina en esta tesis *nivel operativo*, que es el que obtiene la *IU Final* directamente ejecutable. En la *sección 4.1.2.1.* se definen detalladamente los tres tipos de IUs que intervienen en los tres últimos niveles de abstracción, tal y como son concebidos en el marco conceptual propuesto.

La figura 4.1 muestra la estructura propuesta, así como las operaciones que intervienen, las cuales se introducen en detalle en la *sección 4.1.5.1.*

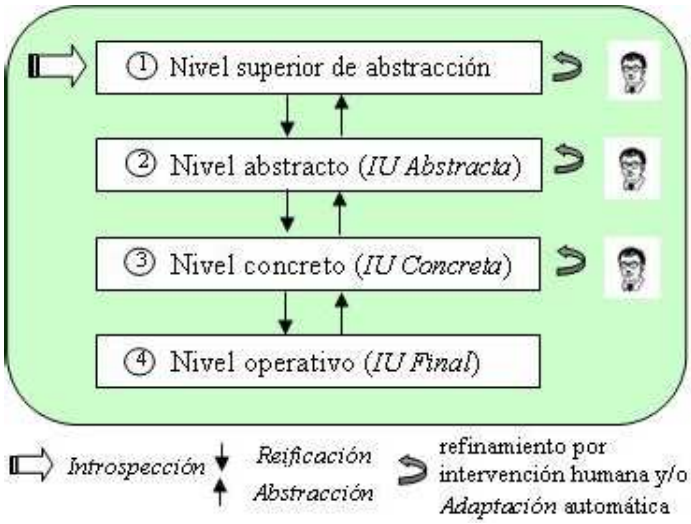


Figura 4.1: Estructura en niveles de abstracción propuesta en el *FPE-C*.

4.1.1.3. Enfoque de modelos compartidos

El marco conceptual propuesto sigue un *enfoque de modelos compartidos* [GMP+98] que consiste en soportar el proceso de generación de la IU mediante la utilización de *repositorios de modelos*. Se entiende por *repositorio de modelos* un área común donde los modelos aportan y comparten conceptos, requisitos y restricciones relativas a su ámbito. En este caso se trata de los conceptos directamente involucrados en la generación de la IU objetivo, con objeto de resolver las restricciones, o bien aplicar las reglas que han de derivar a ciertas deducciones, como paso previo a la generación de la IU. Cabe mencionar

aquí que en el caso de actividades colaborativas, la cantidad de restricciones, deducciones y decisiones automáticas necesarias con objeto de encauzar una actividad grupal se ve considerablemente incrementada. En efecto, cada nuevo evento relativo a un cambio en la situación grupal -comunicado por la plataforma cliente- representa, en la mayoría de los casos, un nuevo estado del grupo de trabajo, el cual debe ser reflejado en el *modelo de grupo*. Estas deducciones se llevarán a cabo a través del uso de *repositorios de modelos*, aplicando las reglas oportunas, tal y como se expone más adelante. Por lo tanto, este enfoque es especialmente adecuado en el caso de entornos colaborativos.

Se puede citar la herramienta basada en modelos Teallach [GMP+98] como ejemplo de utilización del *enfoque de modelos compartidos*. El objetivo de Teallach es el de facilitar el desarrollo sistemático de IUs para bases de datos orientadas a objetos, de manera independiente tanto de la base de datos subyacente como del sistema operativo. Esta herramienta se apoya en el uso de *repositorios de modelos* no sólo para reunir información acerca de los conceptos a compartir en la construcción de la IU, sino también para atacar el problema de la propagación de los cambios sufridos en la IU hacia los modelos subyacentes. En efecto, la falta de mecanismos para propagar cambios es una de las limitaciones detectadas en la literatura [Sze96], tal y como se apunta en el *Capítulo 1* (véase *sección 1.2.*) y en el *Capítulo 3* (véase *sección 3.2.5.*). Los *repositorios de modelos* son utilizados en Teallach, por tanto, como almacén intermedio para conducir la información desde y hacia los modelos correspondientes.

Tal y como se detalla más adelante, el uso de los *repositorios de modelos* que se propone en este marco conceptual tiene también esta doble función. En primer lugar, reunir y recopilar conceptos comunes a los distintos modelos, con objeto de facilitar la explotación de los mismos y el consecuente proceso de propagación de restricciones que subyace a la construcción de IUs. En segundo lugar, facilitar un elemento de soporte donde depositar la información relativa al contexto de uso transmitida desde la plataforma cliente. Su misión en este caso es la de distribuir cada ítem de información al modelo apropiado para proceder a la actualización de los mismos con los cambios producidos durante el uso de la IU, como paso previo a la explotación de unos modelos convenientemente actualizados. Por lo tanto, este enfoque conjuga en armonía con la *visión dicotómica de plasticidad*.

4.1.2. Modelos considerados en el Marco Conceptual

Los modelos que consideramos relevantes, así como la implicación que éstos tienen en el desarrollo de IUs plásticas y *sensibles al grupo* son los que se presentan a continuación, convenientemente clasificados. En particular, los *modelos iniciales* son generalmente especificados manualmente por el desarrollador, con la ayuda de herramientas de modelado

proporcionadas por el sistema que posibilitan su refinamiento. Los *modelos transitorios* son diseños intermedios de la IU generados durante el proceso de obtención de la IU a través de las distintas fases hasta obtener la IU operativa escrita en código fuente. Se requiere de una referencia continua entre los *modelos transitorios* y los *modelos iniciales* para poder llevar a cabo todo el proceso de desarrollo de la IU. Estas referencias quedan especificadas mayoritariamente en las llamadas *reglas de mapeo* [MLV+05].

4.1.2.1. Modelos iniciales

Modelos iniciales se denominan aquellos que constituyen estrictamente una especificación abstracta del sistema interactivo objeto de tratamiento (modelos del nivel superior de abstracción) y los que describen los aspectos relativos a las posibles situaciones en las que se puede hacer uso del sistema (*modelos contextuales*). En las dos subsecciones siguientes se detallan ambos grupos.

4.1.2.1.1. Modelos pertenecientes al nivel superior de abstracción

Contienen las especificaciones propias del sistema interactivo objeto de estudio a un elevado nivel de abstracción, de manera que establecen su dominio aplicativo [CCD+04]. La reutilización de estos modelos sólo es factible, por tanto, al exponer el sistema a distintas situaciones contextuales. Por otro lado, como descripciones de alto nivel del sistema que son, es razonable considerarlos como modelos eminentemente estáticos. No obstante, aunque no reciban una realimentación propiamente dicha, en ocasiones pueden ser objeto de transformaciones o adaptaciones.

En un proceso de derivación puro (consúltese la *sección 4.1.4.1*) estos modelos actuarían desde el principio como conductores del proceso de derivación de la IU para un sistema interactivo en particular.

***Modelo de Tareas* (MTareas)**

Representación estructurada de las tareas que un usuario puede llevar a cabo a través de la IU, así como de las relaciones temporales entre las mismas (por ejemplo, si es factible una ejecución concurrente o por el contrario se requiere una ejecución secuencial). Asimismo, se entiende por *tarea* todo aquello que contribuye a la consecución de un objetivo concreto, y que generalmente contribuye a la modificación del estado del *modelo de dominio* (véase *Capítulo 3*; *sección 3.2.6.2*).

El *modelo de tareas* es el pilar principal en la mayoría de las aproximaciones basadas en modelos, guiando en esos casos la transformación de los modelos hasta la *IU Final*

(desde el punto de vista de un proceso de derivación puro). Es importante describir en detalle las propiedades más relevantes tanto de las tareas como de las acciones en las que éstas se descomponen, así como de todos aquellos detalles de los que se disponga.

En particular, para el caso de actividades colaborativas, el *modelo de tareas* debe ser especialmente acomodado a este tipo de escenarios. Ello conlleva tener en cuenta diversas consideraciones como son: (1) incluir tareas específicas de la actividad grupal, así como también pequeñas acciones destinadas a promover la comunicación y coordinación entre los miembros del grupo durante el desarrollo de la actividad; (2) especificar el carácter individual o grupal de cada tarea; (3) indicar los requisitos iniciales del grupo de trabajo necesarios para llevar a cabo las tareas calificadas de carácter grupal, así como la situación que se alcanza en lo que respecta al grupo una vez realizada (postcondición).

Todas estas consideraciones dan lugar al enunciado de esta primera heurística:

Heurística 4.1 (descripción del modelo de tareas) *Cuanto más detalles se dispongan de cada una de las tareas y de las acciones de que éstas constan, especificando sus propiedades más relevantes -inclusive las relativas a la actividad grupal, si se trata de un entorno colaborativo-, más coherente con la situación en curso, rica y personalizada será la IU resultante, puesto que el proceso de explotación de los modelos se habrá limitado a seguir una lógica minuciosamente especificada.*

Por otra parte, si la descripción de las tareas va acompañada de una descripción de las acciones que determinan un cambio de tarea se facilita la especificación del *modelo del diálogo*. De hecho, una opción válida es describir ambos modelos de forma paralela. Otra opción es la de derivar automáticamente los elementos necesarios para producir la transición entre los distintos estados de la IU. Para conseguirlo es necesario utilizar alguna técnica para decorar el *modelo de tareas* etiquetando convenientemente las transiciones entre las mismas. Este aspecto es el que se recoge en esta segunda heurística:

Heurística 4.2 (descripción transiciones entre tareas - (López-Jaquero, 05)) *La utilización de técnicas de decoración del modelo de tareas, como por ejemplo, mediante el etiquetado de las transiciones entre tareas, permite la derivación automática o semi-automática de los elementos necesarios para producir la transición entre los distintos estados de la IU.*

Modelo de Dominio **(MDominio)**

Representación de la aplicación en general, y en particular de los objetos sobre los que las tareas de usuario actúan e interaccionan (véase *Capítulo 3; sección 3.2.6.1*). En

algunas herramientas este modelo constituye el principal conductor del diseño, como es el caso de Janus [BHKN96].

Entre las notaciones utilizadas para especificar este modelo se puede encontrar desde referencias informales a los objetos de dominio hasta paradigmas estructurados tales como diagramas de entidad/relación o de UML. En particular, los diagramas de clase de UML son los más extendidos, no sólo porque los métodos de diseño basados en el paradigma de orientación a objetos son hoy en día los más utilizados, sino también por su versatilidad a la hora de representar relaciones entre entidades, entre ellas las relaciones de herencia.

El proceso de especificación del *modelo de tareas* y el *modelo de dominio* suele estar asistido por herramientas de modelado proporcionadas por el propio entorno basado en modelos (el MB-UIDE).

Cuanto más detallada sea la descripción de los tipos de datos involucrados tanto en los atributos de las clases como en los métodos, mayor utilidad tendrá este modelo, tanto a través de una explotación manual como automática. En particular, para el caso de actividades colaborativas, el *modelo de dominio* debe incluir conceptos específicos de este tipo de escenarios, como son los actores, los roles, los lugares, los eventos, y cualquier otro aspecto involucrado en el proceso de colaboración. Esta observación es la que se enuncia en esta tercera heurística:

Heurística 4.3 (descripción del modelo de dominio) *Cuanto más detallada esté la descripción de los conceptos de dominio -inclusive las relativas al escenario grupal, si se trata de un entorno colaborativo-, donde, entre otras cosas, es esencial la especificación de los tipos de datos involucrados, una mayor explotación y efectividad podrá obtenerse del mismo en la aplicación de las herramientas automáticas, así como una mayor orientación para el experto humano.*

Modelo de Diálogo (MDiálogo)

Describe la conversación humano-computador, esto es, el "diálogo" entre la IU y el usuario, definiendo en qué momento el usuario puede invocar, seleccionar o especificar diversos comandos, así como cuándo el ordenador puede requerir la intervención del humano para llevar a cabo esas acciones. El *modelo de diálogo* permite establecer el estilo de navegación, así como la definición de los estados de la IU y las correspondientes transiciones entre estados (véase *Capítulo 3; sección 3.2.6.3*).

Tal y como se enuncia en la *Heurística 4.2* anterior, una detallada especificación del *modelo de tareas*, donde la transición entre tareas esté convenientemente documentada,

puede dar lugar a una derivación automática o semi-automática de los elementos responsables de la transición entre los estados de la IU, esto es, del *modelo de diálogo*.

Así, por ejemplo, en AB-UIDE [Lóp05] el *modelo de tareas* está decorado mediante el etiquetado de las transiciones entre tareas con las herramientas abstractas denominadas *canonical abstract prototype* [Con03]. Ello permite que tanto el *modelo de tareas* como *el modelo de diálogo* puedan ser especificados al mismo tiempo.

En particular, para el caso de actividades colaborativas el *modelo de diálogo* debe estar especialmente acomodado a este tipo de escenarios. Ello conlleva tener en cuenta diversas consideraciones como son: (1) incluir transiciones específicas para soportar la actividad grupal, como por ejemplo acciones destinadas a promover la comunicación y coordinación entre los miembros del grupo durante el desarrollo de la actividad. Esto puede dar lugar a un incremento tanto en el número de transiciones posibles como en el número de estados de la IU con respecto a un planteamiento de la actividad estrictamente individual; (2) indicar los requisitos iniciales del trabajo en grupo para llevar a cabo cada transición, así como la situación grupal que se alcanza una vez producidas (postcondición).

Todas estas consideraciones dan lugar al enunciado de la siguiente heurística:

Heurística 4.4 (relaciones entre el modelo de diálogo y el de grupo) *Cuanto más completa esté la descripción del modelo de diálogo, contemplando todas las posibilidades relativas a la dinámica de grupo, convenientemente acompañada de cuantos detalles se dispongan de cada una de las transiciones entre estados, más coherente con la situación grupal y el estado del conocimiento compartido será la IU resultante, de cara a obtener una IU* sensible al grupo*.*

Adicionalmente, los beneficios serán mayores si, además, se especifican las relaciones entre las transiciones entre estados de la IU y las posibles situaciones relativas al grupo de trabajo y el estado de la actividad grupal que se está llevando a cabo (modelo de grupo).

4.1.2.1.2. Modelos contextuales

En general, los *modelos contextuales*, a diferencia de los anteriores, actúan más bien como una base de conocimiento que en algunos casos puede ser reutilizada, no sólo en diferentes procesos de diseño de un mismo sistema, sino también en otros sistemas donde se contemplen condiciones similares. Es el caso de los *modelos de plataforma*, *modelo de usuario* y *modelo espacial*. En otros casos, debido a su naturaleza tremendamente variable en determinados escenarios –escenarios donde se requiere cierta movilidad y escenarios colaborativos-, es indispensable una actualización constante de los mismos a medida que

se va haciendo uso del sistema y el usuario se va encontrando con distintas situaciones contextuales. Es el caso del *modelo de entorno*, y especialmente del *modelo de grupo*, en los cuales no es factible la reutilización. Precisamente es en estos casos que resulta imprescindible el manejo de algún tipo de mecanismo de propagación de cambios con objeto de mantenerlos actualizados.

Los *modelos contextuales* que se proponen en esta tesis se presentan a continuación.

***Modelo de Usuario* (MUsuario)**

Representación de las características y aspectos relacionados con el usuario, tales como el nivel de conocimiento, preferencias, metas, necesidades, etc. (véase *Capítulo 3; sección 3.2.6.4*). En definitiva, se trata de una representación del perfil del usuario, tal y como se describe para la primera componente del contexto de uso en el *Capítulo 2* (véase *sección 2.1.3*).

Dependiendo de si esa caracterización del usuario se limita a un conjunto predefinido de parámetros que responden a un estereotipo establecido en tiempo de diseño, o bien si conforme el usuario interactúa con el sistema –fase de ejecución- se van infiriendo nuevas necesidades y preferencias, las cuales van siendo incorporadas también en el *modelo de la interfaz* mediante un adecuado mecanismo de propagación, se tratará de un sistema *adaptable* y/o *adaptativo* respectivamente, de acuerdo a la distinción existente entre ambos términos (véase *Capítulo 2*; *sección 2.1.2*).

***Modelo de Plataforma* (MPlataforma)**

Expresión explícita de las plataformas objetivo, en términos de los recursos físicos cuantificados y características software (sistema operativo, versión de la máquina virtual de java, en su caso, familia de dispositivos, configuración y perfil en el caso de dispositivos móviles, etc.), tal y como se describe en la segunda componente del contexto de uso en el *Capítulo 2* (véase *sección 2.1.3*).

***Modelo Espacial* (MEspacial)**

Descripción espacial detallada del mundo real, en los sistemas donde se requiera la demanda de información basada en la localización, una de las modalidades más potentes y extendidas hoy en día de personalización de los servicios móviles. Para ser más precisos, en este modelo se especifican los objetos geográficos estáticos que, con objeto de evitar un vacío (gap) semántico, su ordenación por nivel de importancia, y de forma particularizada para cada perfil de usuario puede ser de utilidad en el proceso de obtención de la IU, mediante la aplicación de heurísticas apropiadas. En cualquier caso, la misión de este modelo es la de proporcionar la información espacial necesaria para alimentar un sistema

sensible a la localización, tal y como se describe en el *Capítulo 2* (véase *sección 2.1.3*), como parte de la tercera componente para caracterizar el contexto de uso.

***Modelo de Entorno* (MEntorno)**

Descripción de cualquier condición ambiental (tal como el nivel de luminosidad, el nivel de ruido, las condiciones meteorológicas, o incluso otras particularidades relacionadas por ejemplo con el estado físico del usuario) o temporal (la hora, el día de la semana, etc.), tal y como se describe en la tercera componente del contexto de uso en el *Capítulo 2* (véase *sección 2.1.3*).

Estas consideraciones son específicas de cada dominio de aplicación, y en determinadas aplicaciones su intervención es determinante en la adaptación.

***Modelo de Grupo* (MGrupo)**

Este modelo, a considerar en el caso de entornos colaborativos, consiste en una representación de todos aquellos aspectos relativos al transcurso de la actividad grupal que son relevantes en el desarrollo del trabajo en grupo desde una perspectiva global, y que contribuyen a la construcción de un *conocimiento compartido* y, en consecuencia, a un entendimiento general del estado de la actividad grupal. Este entendimiento, denominado *consciencia de conocimiento compartido*, es esencial para fomentar una colaboración real entre los miembros del grupo de trabajo, previa asimilación por cada uno de ellos. La misión del *modelo de grupo* es la de actuar como repositorio común de los acontecimientos que pueden ejercer una influencia en el desarrollo de la actividad grupal. Se puede calificar por tanto como una memoria del grupo de trabajo o representación del *conocimiento compartido* [CGPO02], tal y como se define en el *Capítulo 2* (véase *sección 2.1.3.*). Se propone su construcción mediante la recopilación de las percepciones individuales de cada miembro del grupo, conforme esta información va siendo comunicada al *servidor de plasticidad* a través de las peticiones explícitas emitidas por cada uno de los miembros del grupo. De hecho, esta es la cuarta componente del contexto de uso (véase *Capítulo 2*; *sección 2.1.3*), considerada en esta tesis componente esencial para la correcta consecución de la meta de grupo.

La información a especificar por este modelo comprende desde las restricciones actuales del grupo, las actividades de sus miembros integrantes, incidencias que afectan en el desarrollo grupal y cualquier otro detalle acerca de cómo se está llevando a cabo la actividad grupal, inclusive cualquier información relativa al conocimiento que del grupo tienen cada uno de los miembros integrantes. El propósito a perseguir es el de proporcionar una visión de conjunto que pueda revertir en el buen funcionamiento de la actividad grupal.

Una posibilidad para modelar este tipo de información, a fin de contemplar las características esenciales de los sistemas colaborativos y facilitar el proceso de diseño y generación de la IU es la notación CIAN-CIAM [Mol07]. Se trata de una herramienta para la especificación y gestión de trabajo cooperativo.

Cabe señalar que la relación entre el *modelo de tareas* y el *modelo de grupo* en entornos colaborativos cobra una importancia primordial en el desarrollo de la actividad grupal. Tanto es así que, además de detallar los aspectos específicos relativos a la actividad grupal en el *modelo de tareas*, se recomienda la especificación explícita de la relación entre ambos modelos. Es por ello que se introduce la siguiente recomendación:

Recomendación 1 (reglas de colaboración) *Con objeto de garantizar que en el proceso de construcción de la IU para entornos colaborativos se le está otorgando la importancia oportuna a la situación relativa al grupo de trabajo y al estado de la actividad grupal –expresados en el modelo de grupo-, es recomendable disponer de una descripción explícita de las relaciones existentes entre las tareas a desarrollar, las restricciones relativas al grupo a considerar en su realización y los conceptos de dominio intrínsecos al grupo, a materializar a través de lo que se denomina en esta tesis reglas de colaboración.*

Más adelante se ofrece una descripción detallada de esta componente, introducida específicamente para entornos colaborativos, las cuales están concebidas para comunicar los requisitos y restricciones del funcionamiento específico del sistema colaborativo objeto de estudio.

4.1.2.2. Modelos transitorios

Los *modelos transitorios* corresponden a las descripciones de la IU a distintos niveles de abstracción, las cuales se van obteniendo a lo largo del proceso de refinamiento –o en su caso, abstracción. Se trata de la *IU Abstracta*, la *IU Concreta* y la *IU Final*. Aunque se trata de conceptos que ya han aparecido en el *Capítulo 3*, a continuación se muestran las definiciones, tal y como son concebidas para el marco conceptual propuesto en la presente tesis.

Definición 4.2 (IU Abstracta): Especificación de alto nivel de abstracción no sólo de los componentes abstractos a formar parte de la *IU Final* y su disposición en la misma –esto es, su estructura estática-, sino también de la manera como la IU evolucionará en el tiempo. Esta descripción genérica es independiente tanto de la plataforma –y por tanto de los elementos de interfaz- como de la modalidad de interacción. □

De hecho, este concepto se corresponde con la definición de *IU genérica* introducida en el *Capítulo 2* (véase *sección 2.1*), así como también con la de *IU Abstracta* introducida en [CCT+02a]. En particular, en este último trabajo se define como *"la expresión canónica de la renderización de los conceptos de dominio y funciones*", la cual es independiente de la modalidad de interacción. Este concepto se corresponde también con las nociones de *Modelo de IU Abstracta* e *IU Abstracta* a los que se hace alusión en el *Capítulo 3*.

La esencia de la *IU Abstracta* está en definir ciertos espacios de interacción (en ocasiones a través de *unidades de presentación*) agrupando subtareas de acuerdo a varios criterios, entre los que se encuentran los patrones estructurales del *modelo de tareas*, el análisis de carga cognitivo y la identificación de relaciones semánticas [LV04].

Una *IU Abstracta* se describe a través de *objetos abstractos de interacción*, los cuales consisten en una abstracción que permite la descripción de objetos de interacción de manera independiente de la modalidad en la cual será representado en el mundo físico. Pueden ser de dos tipos: *contenedores abstractos* y *componentes individuales abstractos* [MLV+05].

Definición 4.3 (IU Concreta): Instancia concreta y suficientemente detallada de la *IU Abstracta*, que es dependiente de la modalidad de interacción, aunque independiente de la plataforma en la cual será representada en el mundo físico. □

Tal y como se introduce en [CCT+02a], una *IU Concreta* convierte una *IU Abstracta* en una expresión dependiente de los *elementos de interfaz*, aunque independiente de los *widgets* finales.

Aunque una *IU Concreta* hace explícito el aspecto final de la IU, todavía se trata de un prototipo o maqueta que es operativa exclusivamente en el entorno de desarrollo, esto es, en el MB-UIDE utilizado.

Una *IU Concreta* se describe a través de *objetos concretos de interacción*, como resultado de la cristalización de los *objetos abstractos de interacción* para una modalidad y condiciones contextuales concretas.

Definición 4.4 (IU Final): IU generada a partir de la *IU Concreta* y expresada en código fuente, en el lenguaje de programación propio de cada plataforma objetivo. Se trata de una versión de la IU preparada para ser en enlazada con el resto de la aplicación, y por lo tanto a punto para ser ejecutada, como IU con la que el usuario final interactuará. □

En la *sección 4.1.5* se presentan en detalle las fases que componen el marco conceptual propuesto. No obstante, conviene avanzar aquí cuáles son estas fases para cada modalidad.

En el caso de seguir un proceso de derivación puro existen cuatro pasos de refinamiento hasta la obtención de IUs, los cuales dan lugar a las sucesivas versiones de la IU, definidas en esta sección. Estas fases son, siguiendo el orden de derivación: *Preparación de los modelos iniciales* (PMI en la figura 4.2 -pág. 182), *Proceso de Rendering Abstracto* (PRA), responsable de la obtención de la *IU Abstracta*, *Proceso de Rendering Concreto* (PRC), encargada de producir la *IU Concreta* e *Implementación* (IMP en la figura 4.2), la cual genera la *IU Final*. En el caso de aplicar un proceso de abstracción puro, comenzando con una implementación concreta de la *IU Final*, se definen tres pasos de inferencia hasta la obtención de los diseños a un alto nivel de abstracción. Las sucesión de las respectivas fases es la siguiente: *Proceso de Inferencia Concreto* (PIC), que obtiene una descripción de la *IU Concreta*, *Proceso de Inferencia Abstracto* (PIA), responsable de obtener la *IU Abstracta* y *Preparación de los modelos iniciales*, para inferir los modelos del nivel superior. Esta última es coincidente en ambas modalidades, a pesar de involucrar pasos distintos. No obstante, tal y como se detalla más adelante, el marco conceptual soporta la combinación de ambas técnicas con absoluta flexibilidad, permitiendo intercalar adecuadamente todas estas fases.

4.1.3. Componentes adicionales del Marco Conceptual

En el FPE-C se proponen una serie de componentes que son comunes a cualquier tipo de escenario, tanto individual como grupal. Adicionalmente, para escenarios de trabajo colaborativo se requieren, además del *modelo de grupo* presentado anteriormente, otro tipo de componentes específicas. En las dos subsecciones siguientes se presentan tanto los componentes generales como los específicos para el trabajo en grupo.

4.1.3.1. Componentes comunes para cualquier tipo de escenario

Para empezar se presentan los componentes comunes tanto para entornos colaborativos como para entornos individuales.

4.1.3.1.1. Reglas de decisión

Estas reglas son útiles para formalizar las restricciones existentes entre distintos aspectos representados a través de los modelos, en concreto de los *modelos iniciales* (*modelos del nivel superior de abstracción* y *modelos contextuales*) semánticamente relacionados entre sí y pertenecientes a un mismo nivel de abstracción.

La propagación de estas restricciones entre los modelos ofrece un soporte para la toma de decisiones, así como también para la inferencia de deducciones a un determinado nivel,

previo al paso de derivación/abstracción. Esta posibilidad toma verdadera relevancia en dos puntos concretos del proceso llevado a cabo en el *servidor de plasticidad*:

1. el momento en que se decide en qué fase iniciar el proceso de producción de una nueva IU, una vez actualizados los modelos.

2. el momento en que se deduce qué tareas son aplicables para unas condiciones contextuales concretas, de todas las previstas en el *modelo de tareas* en un momento determinado. Por supuesto, este tipo de decisiones sólo se producen si se ha decidido iniciar el proceso de derivación/inferencia en una de las dos fases de mayor nivel de abstracción, que son las fases en las que interviene el *modelo de tareas*. Este segundo tipo de deducciones se llevan a cabo una vez haber actualizado los modelos y decidido en qué fase iniciar el proceso, como paso previo a la derivación/inferencia.

Se distingue entre dos grupos de reglas de decisión, justamente para diferenciar estas dos intervenciones: las *reglas de decisión 2* y las *reglas de decisión 1*, respectivamente. La incidencia de estas reglas es, respectivamente, en el *repositorio de modelos 2* y el *repositorio de modelos 1*, tal y como se observa en la figura 4.2 (pág. 182).

Como guía para la materialización de estas reglas, a continuación se exponen una serie de observaciones, formalizadas a través de heurísticas. En particular, es esencial disponer de una información suficientemente detallada de las relaciones entre el *modelo de tareas* y el *modelo de dominio*, relaciones que pueden ser formalizadas a través de las *reglas de decisión*. En particular, en el caso de entornos colaborativos es esencial disponer también de una información suficientemente detallada de las relaciones entre el *modelo de tareas* y el *modelo de* grupo, así como entre el *modelo de diálogo* y el *modelo de grupo*, aspecto que ya ha sido reflejado a través de la *Heurística 4.4* anterior. En muchas ocasiones estas relaciones están embebidas en los propios *modelos de dominio* y/o *tareas*. La opción que se considera más adecuada, y que por ello se propone en esta tesis es la de expresar estas relaciones a través de las *reglas de decisión*.

Estas observaciones dan lugar a las dos heurísticas siguientes:

Heurística 4.5 (relaciones entre el modelo de tareas y el de dominio) *Cuanto más detallada esté la especificación de las relaciones entre las acciones y los elementos de dominio con los que interactúan –por supuesto, provistos con la descripción de los tipos de datos involucrados-, más apropiados serán los objetos abstractos de interacción derivados (paso de derivación) o más apropiada será la deducción de los objetos de dominio que éstos representan (paso de abstracción), pudiéndose garantizar un buen resultado incluso al delegar en las herramientas automáticas.*

Esta heurística puede considerarse como una evolución de la *Heurística 3.2* propuesta por Thevenin y Coutaz en [TC99] (*véase Capítulo 3*; *sección 3.3.1.2*). Además, constituye una generalización de lo que ya se enuncia en [BV94]:

"los tipos de datos de los atributos y métodos, junto con las tareas de interacción a las que están asociados son los que determinan qué objetos concretos de interacción son los más apropiados para cada caso" [BV94].

La extensión de esta heurística al caso de los entornos colaborativos da lugar a la *Heurística 4.6*.

Heurística 4.6 (relaciones entre el modelo de tareas y el de grupo) *Cuanto más detallada esté la especificación de las relaciones entre las acciones -e indirectamente los elementos de dominio (Heurística 4.5)- y las posibles situaciones relativas al grupo de trabajo y el estado de la actividad grupal que se está llevando a cabo (modelo de grupo), más apropiados y acordes con el estado del trabajo en grupo serán los objetos abstractos de interacción derivados (paso de derivación) o más apropiada será la deducción de los objetos de dominio que éstos representan (paso de abstracción), pudiéndose garantizar un buen resultado incluso al delegar en las herramientas automáticas.*

Para hacer uso de las *reglas de decisión* es esencial soportar este proceso a través del uso de *repositorios de modelos*.

Por supuesto, estas reglas son particulares para cada sistema.

4.1.3.1.2. Las reglas de mapeo

Las *reglas de mapeo* permiten representar y almacenar las relaciones que van apareciendo entre unos modelos y otros. Se va construyendo manual o automáticamente, a partir de las sucesivas derivaciones o inferencias, permitiendo al diseñador observar cómo se ha llevado a cabo el proceso de derivación/inferencia. Este proceso puede incluso ser modificarlo, siempre y cuando se proporcione una herramienta visual a tal efecto.

Posibles ejemplos de *reglas de mapeo* son los siguientes: en qué contenedor se visualizará una tarea concreta; qué elemento del dominio manipula cada tarea; qué conjunto de *objetos concretos de interacción* representa cada conjunto de *objetos abstractos de interacción*.

Estas reglas pueden implicar modelos de un mismo nivel de interacción, o bien modelos pertenecientes a dos niveles sucesivos de abstracción, como son la *IU Abstracta* con la *IU*

Concreta -o viceversa en el caso de un proceso de abstracción-, o elementos de los *modelos de dominio* y el *de tareas* con la *IU Abstracta*.

Los mapeos permiten mantener la trazabilidad del sistema, información que resulta de gran utilidad para disponer de las relaciones entre los modelos de la IU que dieron lugar a la *IU Concreta* producida con anterioridad, y poder reutilizarla para una nueva derivación.

Así, por ejemplo, si se reduce el área de presentación en pantalla, y la IU intenta acomodarse a esta nueva situación será necesario conocer qué tipos de datos están siendo manejados. Así, si en la aplicación un valor lógico se estaba presentando en forma de dos botones de radio (*radioButton*), podría ser presentado usando una casilla de verificación para ahorrar espacio. No obstante, para ello es necesario saber que es un valor lógico lo que se está representando en la IU, y que por lo tanto sólo se necesita un control capaz de alternar entre dos valores. Este tipo de información es la que se construye y se registra a través de las *reglas de mapeo*.

Las relaciones existentes entre los modelos, especificadas a través de las *reglas de mapeo*, describen por tanto la estructura interna y relaciones entre sus componentes, es decir, definen la arquitectura interna de la IU [Lóp05]. Según Eisenstein et al. en [EVP01], *"estas relaciones son las responsables del comportamiento interactivo de la IU."*

En el caso de aplicaciones móviles cobran vital importancia las relaciones entre el *modelo de plataforma* y el de *presentación*, concretamente la *IU Concreta* [EVP01], puesto que describen cómo las restricciones impuestas por la plataforma influyen ya desde el nivel concreto en la apariencia visual de la IU (proceso de reificación).

En ocasiones, estas relaciones entre modelos se formalizan a través de reglas (las *reglas de mapeo*), mientras que en otros casos se utiliza un modelo propio: el denominado *modelo de mapping* –encargado de reunir las relaciones establecidas entre modelos-, utilizando su propio lenguaje de modelado. A veces estas especificaciones están diluidas en la descripción de otros modelos, o incluso llegan a confundirse con ellos. Sin ir más lejos, en determinados casos el *modelo de diálogo* puede considerarse como una especificación de las relaciones entre el *modelo de tareas* y el *modelo de presentación*, como es el caso del lenguaje *Mastermind Dialog Language* (MDL) [SSC+96]. Otro ejemplo es el caso de la notación CTT [Pat99], que incluye relaciones entre tareas donde se recogen mapeos tarea-diálogo.

Las primeras propuestas de *reglas de mapeo* encontradas en la literatura son las propuestas en [Van95] para IUs gráficas y las propuestas en [Sut97] y [Ber94] para IUs

multimedia e IUs multi-modales.

Cabe destacar aquí el trabajo presentado en [MLV+05], donde se introduce una propuesta de solución general al *problema del mapeo* entre modelos basada en *patrones*[2], con el objetivo de ayudar al desarrollador en el proceso de modelado. En este contexto se entiende por *patrón* una abstracción de una agrupación de entidades –artefactos utilizados en la construcción de modelos- con una elevada probabilidad de resultar útiles por su reiterada aparición en entornos MB-UIDE. Igual que cualquier otro tipo de patrones[3], éstos se identifican como resultado de la experiencia, en este caso en la construcción de *modelos de interfaz* y de la consecuente observación de las relaciones establecidas entre los modelos. En particular, se ofrece un conjunto de relaciones predefinidas que explicitan las relaciones entre el *modelo de dominio* y el resto de modelos de la interfaz, algunas de ellas ya identificadas por Puerta en [PE99].

Adicionalmente, en [MLV+05] se propone recoger este tipo de patrones utilizando usiXML [LVM+04] (véase *Capítulo 3*; *sección 3.1.3*), como lenguaje uniforme entre modelos heterogéneos e IdealXML como herramienta que facilita al diseñador la especificación de relaciones, con el objetivo de proporcionar un marco de trabajo integrado y uniforme (formalismo único) para la edición de modelos y relaciones de mapeo entre modelos. UsiXML es definido como un conjunto de XML schemas, donde cada uno de estos schemas corresponde a un modelo. Además, proporciona explícitamente un *modelo de mapping*.

En el marco conceptual propuesto se reflejan las *reglas de mapeo* utilizando tres grupos de reglas, uno para cada uno de los niveles (abstracto, concreto y operativo), a fin de especificar la manera en que se han ido relacionando los elementos en cada una de las fases de derivación o inferencia.

4.1.3.1.3. Repositorios de modelos

Como ya se ha comentado anteriormente, los *repositorios de modelos* son áreas comunes intermedias donde los modelos participantes en cada fase aportan y comparten información y conceptos relativos a su ámbito, como soporte para su manejo y actualización, los cuales están involucrados en la generación de la IU objetivo. Esto permite sacar provecho de que muchos de los conceptos manipulados por los modelos son análogos entre sí o aparecen en

[2]Un patrón afronta un problema de diseño recurrente que surge en situaciones de diseño específicas, proporcionando una solución al mismo.

[3]Este concepto no debe confundirse con el de *patrón de interacción*, como parte de la componente denominada *Directivas de interacción*, detallada más adelante. Estos últimos son sólo descripciones en lenguaje natural que no han sido modeladas.

más de un modelo. Este soporte es especialmente valioso en el caso de entornos colaborativos, donde el número de consideraciones, restricciones y, en consecuencia, la necesidad de llevar a cabo deducciones y decisiones automáticas cobra aún más importancia.

El marco conceptual propuesto plantea la necesidad de utilizar un *repositorio de modelos* para las dos fases intermedias (*Proceso de Rendering Abstracto* y *Proceso de Rendering Concreto* -véase *sección 4.1.5.1*-) en un proceso de derivación y las dos últimas fases (*Proceso de Inferencia Abstracto* y *Proceso de Inferencia Concreto* -véase *sección 4.1.5.2*-) en un proceso de abstracción, con objeto de compartir conceptos entre cada grupo de modelos involucrados en cada una de ellas.

Sobre los *repositorios de modelos* actúan las *reglas de decisión*, con objeto de llevar a cabo los/as sucesivos/as refinamientos/inferencias sobre la IU.

4.1.3.1.4. Reglas de adaptación

Descripción de las reglas aplicables sobre un *modelo transitorio* no final (*IU Abstracta* e *IU Concreta*) ya generado previamente, con el propósito de obtener una nueva versión del mismo en el mismo nivel de abstracción, esto es, otra descripción de la *IU Abstracta* o *Concreta*, acomodada a una situación contextual distinta. Este proceso de adaptación evita, cuando así se cree conveniente, tener que iniciar desde cero, esto es, desde el nivel más abstracto en un proceso de derivación, o desde el nivel operativo en un proceso de abstracción, el proceso de generación del *modelo transitorio* en cuestión para cada *cambio contextual*.

De hecho, lo que aquí se presenta como un paso de adaptación de los *modelos transitorios* corresponde a la *operación de traducción* entre contextos de uso descrita en el *marco de referencia unificado* introducido en el *Capítulo 3* (véase *sección 3.3.2*). Esta capacidad de adaptación de los modelos constituye un ejemplo de reutilización de diseños ya existentes de la IU en distintos niveles de abstracción.

Tal y como se propone en el *marco de referencia unificado*, esta *operación de traducción* también puede ser aplicada en el nivel superior de abstracción, especialmente sobre el *modelo de tareas*. Es el caso en el que, dada una nueva situación contextual que configura unas condiciones de trabajo distintas, ya sean más restrictivas –e. g. por el hecho de haber migrado a un dispositivo más limitado que hace que el conjunto de tareas factibles de realizar se vea reducido-, o a la inversa –ampliación de las posibles tareas a realizar-, el *modelo de tareas* requiere una operación de poda o expansión, respectivamente, para su adecuación a la nueva situación contextual.

En efecto, en ciertas herramientas basadas en modelos el *modelo de tareas* está construido de tal manera que para cada tarea se detalla en qué plataformas es factible su realización (algunas tareas no son susceptibles de ser realizadas en algunos dispositivos, como por ejemplo la escritura de un documento en un teléfono móvil), o bien en qué tipo de entornos o circunstancias son apropiadas cada una de ellas (por ejemplo, escuchar música no es compatible con la actividad de asistir a una clase o conferencia, ni tampoco en un entorno con mucho ruido de fondo). Este tipo de especificaciones puede ser ampliado para indicar también ante qué necesidades y preferencias del usuario (por ejemplo, si ha alcanzado cierto nivel de experiencia no es necesario que se le muestre la posibilidad de consultar información adicional) o incluso en qué situaciones de grupo tiene sentido llevar a cabo cada tarea (así, se puede dar por finalizada una actividad grupal si las tareas a desempeñar por cada uno de los integrantes del grupo han sido ya concluidas; otro ejemplo es que determinadas tareas no son factibles de realizar si no se dispone del suficiente número de personas disponibles para su consecución; únicamente deben estar disponibles en la IU aquellas tareas que pueden ser llevadas a cabo).

En estos casos en los que las tareas están decoradas con información contextual, la operación de "poda" o extensión de dicho modelo a través de las *reglas de adaptación*, permite tomar en consideración todos estos aspectos ya en el nivel superior de abstracción. En definitiva, se propone extender las situaciones en las que puede requerirse adaptar explícitamente el *modelo de tareas* a cualquier cambio contextual (los atributos del contexto considerados en esta tesis son: usuario, entorno, plataforma, grupo, tarea). Se puede decir que la información aportada relativa a las tareas está restringida a diversos factores contextuales, por lo que éstos empiezan a tener repercusión ya desde el nivel superior de abstracción.

Con objeto de que el marco conceptual propuesto contemple también esta posibilidad, y efectivamente pueda representar también las herramientas basadas en modelos que consideran esta opción, se incluye la posibilidad de adaptar no sólo el *modelo de tareas*, sino también los otros dos modelos del nivel superior (*modelo de dominio* y de *diálogo*) utilizando estas reglas. En la figura 4.2 (pág. 182) queda reflejado a través de las siguientes conexiones entre componentes: (1) una relación entre las *reglas de adaptación* y las herramientas automáticas correspondientes al nivel superior de abstracción; (2) las flechas de retorno desde estas herramientas automáticas hacia esos tres modelos; (3) además, como las *reglas de adaptación* podrían incluso modificarse a sí mismas para reflejar su propio refinamiento de acuerdo a nuevos datos inferidos, se añade una nueva dependencia representada por una flecha de color púrpura entre el *repositorio de modelos 2* y las *reglas de adaptación*; y por último (4) la conexión –flecha ascendente de color rojo- entre el *reposi-*

torio de modelos 2 y el *repositorio de modelos 1*, que indica la realimentación relativa al contexto de uso (aspectos relativos a la plataforma, entorno, usuario o estado del trabajo en grupo), procedente de la descripción recibida de la plataforma cliente, con objeto de que el *modelo de tareas*, y el resto de modelos del primer nivel de abstracción, en su caso, pueda ser podado, extendido o simplemente adaptado a la nueva situación contextual.

4.1.3.1.5. Reglas de transformación

Estas reglas describen el proceso de transformación en vertical (derivación o inferencia) de unos modelos en otros, esto es, entre distintos niveles de abstracción. Se trata de las reglas que dirigen la transformación de las distintas representaciones de la IU a distintos niveles de abstracción. En un proceso de derivación especifican cómo derivar los diseños desde niveles abstractos a niveles concretos. En cambio, para un proceso de ingeniería inversa, incorporan los detalles necesarios para lograr la inferencia de diseños abstractos a partir de otros más concretos.

En el marco conceptual se especifican estas transformaciones en tres niveles de transformación. Se proponen, por tanto, tres grupos de *reglas de transformación* distintos, independientemente de si se aplica derivación o inferencia: (1) de los modelos del nivel superior de abstracción en la *IU Abstracta* (*Proceso de Rendering Abstracto*) o viceversa (*Preparación de los Modelos Iniciales*); (2) de la *IU Abstracta* en la *IU Concreta* (*Proceso de Rendering Concreto*) o viceversa (*Proceso de Inferencia Abstracta*) y (3) de la *IU Concreta* en la *IU Final* (*Implementación*) o viceversa (*Proceso de Inferencia Concreta*).

En el primer caso se trata de automatizar la derivación de la *IU Abstracta* a partir de una descripción tanto de las tareas de interacción como del *modelo de dominio* de la aplicación. En el segundo caso se trata de determinar qué componente concreta o conjunto de componentes concretas de interfaz (*objetos concretos de interacción*) representarán la funcionalidad descrita por cada componente o conjunto de componentes abstractas (*objetos abstractos de interacción*), para ir dando forma a la IU, de acuerdo a toda la información contextual recogida en los *modelos contextuales*. En otras palabras, especifican cómo se transforma la *IU Abstracta* en *IU Concreta*. Por último, en el caso de la *IU Final* se trata de determinar el conjunto de componentes de interfaz finales, propio de la plataforma destino, adecuados a los componentes concretos (*objetos concretos de interacción*).

A continuación se presentan en detalle cada grupo de *reglas de transformación*.

Reglas de transformación 1 (derivación de la IU Abstracta e inferencia de los modelos del nivel superior)

Estas reglas describen la correspondencia entre los conceptos del dominio y las tareas involucradas (*modelos de dominio y tareas*) con las estructuras proporcionadas en el MB-UIDE para modelar contenedores de información y unidades elementales asociadas a esos conceptos (en general, referidos como *objetos abstractos de interacción*), todo ello a un nivel apropiado de granularidad.

Se trata de las reglas que ayudan en la selección de qué *objetos abstractos de interacción* se comprometen a soportar cada uno de los conceptos de dominio, tareas y situaciones de la actividad grupal en el caso de entornos colaborativos (paso de *derivación*), así como también en la deducción de qué objetos de dominio son representados por cada uno de los *objetos abstractos de interacción* (paso de *abstracción* hacia la obtención de los modelos del nivel superior).

Reglas de transformación 2 (derivación de la IU Concreta e inferencia de la IU Abstracta)

Describen la correspondencia entre los *objetos abstractos de interacción* y los *elementos de interfaz* (generalmente referidos como *objetos concretos de interacción*) [VB93].

Estas *reglas de transformación* tienen como objetivo ayudar en la selección de qué conjunto de *objetos concretos de interacción* es capaz de renderizar cada conjunto de *objetos abstractos de interacción*, utilizando básicamente información acerca del contexto (paso de *reificación*), así como también en la deducción de qué *objetos abstractos de interacción* son representados por cada uno de los *objetos concretos de interacción* (paso de *abstracción* hacia la obtención de la *IU Abstracta*).

Reglas de transformación 3 (obtención de la IU Final e inferencia de la IU Concreta)

Por lo que respecta al proceso de derivación de la IU, este tercer grupo de *reglas de transformación* describe la traducción de los *elementos de interfaz* (*objetos concretos de interacción*) en los componentes de interfaz finales, de acuerdo a la especificación de la plataforma objetivo (contenida en el *modelo de plataforma*), esto es, teniendo en cuenta sus recursos físicos, que en ocasiones incluso son variables (e.g. capacidad de memoria disponible de un móvil).

En ocasiones estas reglas se basan en la resolución de restricciones entre el *coste de renderización* [TC99] de los *elementos de interfaz*, entendido como la cantidad de recursos físicos requeridos para su instanciación, y la disponibilidad de recursos físicos de la plataforma objetivo [TC99].

Otras veces estas reglas se incluyen implícitamente en los generadores automáticos de código (generalmente denominados *visualizadores o renderers*).

En el caso de aplicar un proceso de ingeniería inversa, este tercer grupo de *reglas de transformación* ayuda en la deducción de qué *objetos concretos de interacción* representan cada grupo de *widgets* que aparecen en la *IU Final*.

En cuanto a la implementación de las reglas de transformación, se puede mencionar el ejemplo concreto de la herramienta AB-UIDE [Lóp05]. En esta herramienta, que utiliza un proceso de derivación puro, tanto las *reglas de transformación* como las *reglas de adaptación* son implementadas siguiendo un modelo basado en la transformación de grafos mediante técnicas de reescritura condicional [LVM$^+$04]. Por otro lado, todo el proceso de transformación es soportado por usiXML [LV04], que además ha sido enriquecido para dirigir el propósito de la plasticidad de un modo más apropiado.

4.1.3.1.6. Criterios de usabilidad

En esta componente se propone definir no sólo el conjunto de propiedades de usabilidad predefinidas en la fase de diseño, a preservar a lo largo de todo el proceso de adaptación, sino también ciertos criterios de priorización. Se trata de unas reglas que dirigen el proceso de adaptación, ejerciendo la suficiente influencia como para priorizar, en beneficio de los criterios que se quieran hacer prevalecer, las adaptaciones consideradas más adecuadas en situaciones donde más de una *regla de adaptación* puede ser aplicada, así como restringir, si es necesario, la evolución de la adaptación, haciendo prevalecer la usabilidad por encima de aquélla. Es importante destacar que, aunque idealmente todos los criterios de usabilidad son importantes, a menudo darle más relevancia a uno en concreto conduce a la reducción de otro, por lo que de alguna manera debe estar especificado cuál es el que se considera más prioritario en cada caso. En definitiva, se trata de tener cierto control acerca de la adaptación de la IU, con objeto de guiar la selección entre distintas posibilidades.

El ejemplo que mejor ilustra este aspecto es el que se presenta en [Lóp05], y que se muestra a continuación. Así, cuando una IU sufre un cambio en el tamaño de la pantalla donde está siendo visualizada, es necesario evaluar las posibles adaptaciones de la misma al nuevo tamaño. Por ejemplo, hacer prevalecer el criterio de visibilidad en este caso puede ser contradictorio con el de maximizar el criterio de accesibilidad. Para ser más precisos, si se desea mantener unos tamaños de fuente suficientemente grandes para mejorar la IU para personas con dificultades de visión, no siempre será compatible con mantener la visibilidad de todos los elementos necesarios para realizar la tarea actual.

En particular, esta componente recibe el nombre de *compromiso de usabilidad* en la herramienta AB-UIDE, puesto que describe el peso e influencia que cada criterio de usabilidad debe tener en la generación de la IU. Concretamente, para su implementación se ha utilizado una variante del *Goal-oriented Requirement Language*[4] [Yu04], basado en la notación I* [Yu97] y el entorno NFR [CNY96]. Esta opción permite al diseñador tanto la captura de los requisitos del *compromiso de usabilidad* como su documentación.

El contexto de uso en el que se presente este tipo de situaciones (entre otras consideraciones, las preferencias del usuario) ejerce un papel primordial en la decisión a tomar, y por ello debe dársele la importancia que merece. El conjunto de propiedades y criterios de usabilidad es, por supuesto, específico de cada sistema interactivo. Además, deben ser particularizados para cada plataforma.

No obstante, no es suficiente con especificar este *compromiso de usabilidad*. Es necesario, adicionalmente, diseñar algún mecanismo que permita evaluar en qué grado dichos criterios se mantienen, pudiendo por tanto determinar si se está cumpliendo el compromiso de usabilidad o no, y de ese modo obrar en consecuencia. Se trata de un mecanismo de control y regulación de los mismos. Para realizar esta evaluación es necesario proponer métricas de usabilidad adecuadas que permitan conocer qué criterios se dan y en qué medida durante el proceso de explotación de los modelos. En particular, se aplican las métricas de diseño propuestas en [CL99] para medir parte de los criterios de usabilidad planteados para un sistema determinado. Su especificación posibilita el modelado de los criterios que el sistema debe seguir para elegir entre las distintas posibilidades de adaptación que se presentan, enfocadas a obtener el máximo valor posible de usabilidad. Por supuesto, en este caso también las métricas de usabilidad deben ser particularizadas para cada plataforma.

Precisamente, esta componente, que actúa en todo momento tratando no sólo de evitar la degradación de la usabilidad, sino de maximizarla, constituye uno de los artefactos necesarios para que las IUs obtenidas del proceso de desarrollo puedan ser calificadas de IUs plásticas, y no meramente multi-contextuales. Su integración en un *Motor de Plasticidad Explícita* es, por lo tanto, imprescindible. En este sentido, nuestro marco extiende el *marco de referencia unificado* (el CAMELEON Reference Framework –véase *Capítulo 3*; *sección 3.3.5*), al proporcionar un mecanismo que vela por la preservación de la usabilidad en cada paso.

Los criterios de usabilidad más ampliamente utilizados se describen en [IFI97], [GWC+00], [CCT01b] y [MS97].

[4] http://www.cs.toronto.edu/km/GRL/

4.1.3.1.7. Directivas de interacción

Esta componente hace referencia a una amplia variedad de modalidades para asistir el diseño de la IU. Agrupa cualquier tipo de guía de estilo [Shn92], [IBM93], [SM86], heurística, criterio ergonómico o *patrón de interacción* [Wel04], [Tid02], [DLH02], [MLML03] que recoja y exprese la experiencia acumulada por los diseñadores de IUs, haciendo posible su reutilización. Intervienen en las dos últimas fases (*Proceso de Rendering Concreto* e *Implementación*) en un proceso de derivación, y en las dos primeras fases en un proceso de abstracción (*Proceso de Inferencia Concreto* y *Proceso de Inferencia Abstracto*), con objeto de guiar la transformación de los *objetos abstractos de interacción* en *objetos concretos de interacción*, y de éstos en los *widgets* finales, y a la inversa si se aplica ingeniería inversa, de acuerdo a las *restricciones de tiempo real* (véase *Capítulo 2*; *sección 2.1.2*), a fin de preservar la usabilidad.

No obstante, aunque estos criterios pueden ser de ayuda, son de difícil aplicación directa por la falta de una estructuración clara [MLLG06].

La herramienta clásica por excelencia que incluyó esta iniciativa es la herramienta TRIDENT (Tools foR an Interactive Development ENvironmenT) [BHL$^+$95] (véase *Capítulo 3*; *sección 3.5*), proporcionando una base de conocimiento en cuanto a directrices de diseño que ofrecen al diseñador un conjunto de componentes de presentación apropiadas, a partir de las cuales poder seleccionar las más convenientes en cada caso. No obstante, tal y como se expone en [Van95], para resolver un problema particular, por concreto que éste sea, no es suficiente con un único enfoque que involucre una única técnica basada en conocimiento. Estos autores proponen considerar un enfoque multi-estrategia.

Otros trabajos que incorporan este tipo de directrices de usabilidad son [VB93] y [VB99]. Como directivas de usabilidad específicas para computación móvil podemos citar [BFO02], [BC99], [BFJ$^+$01] que contiene algunos principios relacionados con usabilidad móvil, o [Rot02] donde proponen patrones de interacción móvil.

4.1.3.2. Componentes específicas para escenarios de trabajo en grupo

Hasta aquí se han presentado los componentes comunes para entornos individuales y colaborativos. A continuación se presentan los componentes específicos para entornos colaborativos.

4.1.3.2.1. Reglas de colaboración

Descripción de las reglas específicas que gobiernan el comportamiento colaborativo del sistema objeto de estudio desde una perspectiva global, plasmando los requisitos y restricciones de su funcionamiento. En efecto, la mayoría de reglas de negocio involucran restricciones acerca de (1) las personas (generalmente referidas a través de los roles) que pueden participar en un contexto determinado, o que pueden interaccionar con otras personas o cosas en ciertos lugares; (2) los eventos –descripción de una interacción entre personas, lugares y cosas- que pueden ocurrir; (3) el lugar donde éstos se pueden producir; (4) los conceptos involucrados, así como la manera en que éstos están agrupados, clasificados y ensamblados. Sin unas directrices que regulen los procesos de negocio, el funcionamiento puede ser totalmente imprevisible, y por tanto propenso a error.

Estas reglas deben aplicarse en el momento en que se le comunica al *servidor de plasticidad* un evento de colaboración (información relevante en el transcurso de la actividad grupal) por parte de uno de los miembros del grupo de trabajo. En función del evento producido se trata de deducir sus repercusiones con objeto de inferir cuál es el patrón de activación del *Motor de Plasticidad Explícita* que conviene poner en práctica (estos patrones se describen en la *sección 4.1.5.2.*). Las *reglas de colaboración* juegan aquí un papel esencial que queda plasmado en la figura 4.2 mediante su conexión con el *repositorio de modelos 2*. Por lo tanto, estas reglas refuerzan a las *reglas de decisión 2* en este momento, haciendo que entren en consideración todo tipo de restricciones, entre ellas las relativas al grupo. Igualmente, sirven de refuerzo también a las *reglas de decisión 1*, tal y como se comenta a continuación.

En efecto, ejercen un papel determinante en el nivel abstracto, en la decisión de la siguiente tarea a realizar, con objeto de que se tenga en cuenta también la situación grupal. Al mismo tiempo ayudan a la deducción del nuevo estado de grupo que se alcanza tras haber analizado las repercusiones del evento de colaboración comunicado al *servidor de plasticidad*. Por supuesto, estas decisiones sólo pueden tomarse teniendo en cuenta el estado de la situación grupal, reflejada en el *modelo de grupo*, la cual deberá recopilarse en ese momento en el *repositorio de modelos 1*. A su vez, la nueva situación grupal inferida en el *repositorio de modelos 1* debe ser también anotada en el *modelo de grupo*. Por lo tanto, la comunicación de un evento de colaboración que comporta la generación de una nueva IU al nivel abstracto desencadena un proceso de realimentación entre el *modelo de grupo* y el *repositorio de modelos 1*. A continuación se introduce un ejemplo para ilustrar esta realimentación:

En una empresa de la construcción (véase Caso de Estudio "Controlador de Obras"; Capítulo 7 - sección 7.1.) los distintos empleados notifican al servidor de plasticidad que han finalizado su agenda para la jornada. En el momento en que un empleado en particular

notifica esa circunstancia pueden darse dos posibles situaciones: (a) se le da el visto bueno para que finalice la jornada; o (b) como existen tareas canceladas pendientes de llevar a cabo, y en esos momentos es factible su realización, se le asigna una de esas tareas al empleado en cuestión.

Como resultado de esa inferencia se alcanza un nuevo estado grupal (un cierto trabajador ha finalizado la jornada, o bien se le ha asignado una tarea pendiente) suficientemente relevante como para ser plasmado en el modelo de grupo.

Este ejemplo, que corresponde al escenario 3 (finalización de la agenda de la jornada) descrito en el *Capítulo 7* (véase *Capítulo 7*; *sección 7.1.3.2.1.*), ilustra el hecho de que para determinados eventos de colaboración no es suficiente con actualizar el *modelo de grupo* en una sola ocasión –conexión con el *repositorio de modelos 2* en la figura 4.2 a través de la operación de *introspección por variación*. Si ese evento comporta iniciar la producción de una nueva IU en el nivel abstracto, un mismo evento de colaboración puede desencadenar una segunda anotación sobre el *modelo de grupo* (la inferida en el servidor de acuerdo a las restricciones y situación actual).

Estas situaciones quedan reflejadas en la figura 4.2 de tres formas: (1) la conexión de las *reglas de colaboración* con el *repositorio de modelos 1* y *2*; (2) la conexión bidireccional entre el *modelo de grupo* y el *repositorio de modelos 1* (posible realimentación mencionada arriba); y (3) conexión *repositorio de modelos 2* y *modelo de grupo*; y (4) la conexión entre el *repositorio de modelos 2* y el *repositorio de modelos 1* (flecha ascendente de color rojo) en los casos en que la información contextual debe ser recopilada también en el *repositorio de modelos 1*, debido a que por la trascendencia del evento comunicado se requiere activar el proceso de generación de una nueva IU en el nivel abstracto. Tal y como se observa, por tanto, estas reglas proporcionan, en definitiva, un soporte para la toma de decisiones relativas a la actividad grupal, enfocando su desarrollo hacia la consecución de la meta de grupo, tratando de fomentar la comunicación y coordinación entre sus miembros.

Los distintos patrones de activación del *Motor de Plasticidad Explícita* a que pueden conducir las *reglas de colaboración*, dependiendo de la trascendencia del evento de colaboración y de la situación grupal del momento son los siguientes: (1) el más simple: única y exclusivamente registrar el evento producido en el *modelo de grupo*, con objeto de actualizar el *conocimiento compartido* y con ello mantener informado al *servidor de plasticidad*, como única repercusión (no se requiere en este caso construir una nueva IU); (2) registrar el evento y activar el proceso de generación de una nueva *IU Abstracta*, ya sea adaptando una versión previa –aplicación de las *reglas de adaptación*-, o bien derivando –o aplicando *abstracción*, en su caso- una nueva *IU Abstracta* teniendo en cuenta las circunstancias

globales del grupo –aplicación de las *reglas de transformación*-; y (3) registrar el evento y activar la adaptación del *modelo de tareas* (fase de *Preparación de los Modelos Iniciales*) de acuerdo a la nueva situación contextual, iniciando así el proceso de derivación de la IU desde el primer nivel de abstracción. Es en estos dos últimos casos que el *repositorio de modelos 1* se realimenta del *repositorio de modelos 2*. Se presenta una explicación completa de todos estos patrones de activación en la *sección 4.1.5.2*.

Sea cual sea el camino y la decisión tomada como consecuencia de este paso de inferencia, de alguna manera debe quedar reflejada esa *consciencia de grupo* en la IU en vías de construcción, que de ese modo cumplirá con la característica de ser una *IU sensible al grupo*. Sólo mediante la entrega al cliente de una IU personalizada a la situación actual de grupo y a las circunstancias e inferencias deducidas se conseguirá ir transmitiendo a los integrantes del grupo de trabajo ese conocimiento global, reunido y explotado en el *servidor de plasticidad*, con objeto de alcanzar y difundir un entendimiento del *conocimiento compartido* (la *consciencia de conocimiento compartido* definida en el *Capítulo 2*; *sección 2.1.3.*) entre todos los miembros del grupo. Sin esta difusión, el entendimiento se perdería en el *servidor de plasticidad*, resultando en vano todo el esfuerzo realizado.

En resumen, podemos concluir que las *reglas de colaboración* ejercen dos funciones principales:

1. inferir la dinámica social del grupo; es por ello que intervienen en (a) el proceso de activación del *Motor de Plasticidad Explícita*; y en (b) el proceso de inferencia de la nueva situación grupal, el cual puede llevar a la deducción de ciertas propiedades globales relativas a la situación en curso del grupo, así como a la toma de decisiones grupales; y

2. intervenir en lo que respecta a los aspectos colaborativos en el proceso de refinamiento -o abstracción, en su caso- de la IU en sus fases iniciales, con objeto de plasmar desde un principio la influencia del *conocimiento compartido* en la generación de la IU. Fruto de la explotación de ese *conocimiento compartido*, representado a través del *modelo de grupo*, se genera una IU personalizada a la situación en la que se encuentra el grupo (IU *sensible al grupo*), a ser transmitida a los propios miembros integrantes.

En particular, los *patrones de negocio*, también conocidos como patrones de análisis [Fow96], se centran en crear modelos de objetos que claramente comunican los requisitos y describen el proceso de negocio, así como las reglas subyacentes al sistema. La clave para modelar procesos de negocio no está en definir los pasos del proceso, sino en centrarse en las

relaciones existentes entre las personas, las cosas, los lugares y los eventos involucrados en el proceso, tal y como es mencionado arriba. En esta línea, Nicola et al. en [NAMA01] reunió y modeló estas relaciones a través de doce patrones. Puesto que estos patrones describen las colaboraciones involucradas en procesos de negocio típicos se les denomina *patrones de colaboración*. Estos patrones son capaces de modelar cualquier dominio de negocio. En el caso de utilizarlos como mecanismo para implementar las *reglas de colaboración* se requeriría de alguna metodología que permitiera transformar esas especificaciones del funcionamiento del sistema en reglas a ser testeadas automáticamente.

La incorporación de esta componente es especialmente recomendada en entornos colaborativos, tal y como se expresa en la *Recomendación 1*, introducida anteriormente.

Como es natural, estas reglas deben ser particularizadas a las necesidades, restricciones de funcionamiento y peculiaridades de cada sistema colaborativo.

4.1.3.2.2. Directrices de colaboración

Esta componente propone materializar las directrices de colaboración más destacadas, a fin de fomentar la explotación y difusión del *conocimiento compartido* y de las deducciones que de él se derivan, a plasmar a través de la IU y de las tareas que en ella se representan, la cual podría entonces ser considerada como una *IU sensible al grupo*. Se trata de las directrices que recogen la experiencia acumulada de los diseñadores e investigadores en el campo de *groupware*, cuya meta principal es la de fomentar la participación entre los miembros del grupo de trabajo y contribuir a una colaboración real.

Tal y como se presenta en el *Capítulo 1* (véase *sección 1.2.*), un reconocimiento de la dinámica social donde la actividad grupal se desarrolla es esencial para alcanzar el éxito de un trabajo en grupo, el cual debe ser concebido como un conjunto organizado de actividades coherentes con buenas estrategias de comunicación, cooperación y coordinación entre los miembros del grupo [AGOP05]. Es en esta línea que los diseñadores de *groupware* han incluido aspectos relacionados con la *consciencia de grupo* –entendimiento colaborativo, definida en detalle en el *Capítulo 2*-, destacando la importancia de soportarla. Con objeto de darle el tratamiento y trascendencia oportunos, debe ser apropiadamente capturada, reunida, representada, explotada y finalmente distribuida. Es aquí donde entran en juego los *mecanismos de consciencia de grupo* [CGPO02] (definidos en el *Capítulo 2* -véase *sección 2.1.3.*-) y, en concreto, los *mecanismos globales de consciencia de grupo* (véase *Capítulo 2*; *sección 2.1.3*), con el objetivo de (1) reunir cada una de las percepciones individuales de cada uno de los miembros recibidas a través de las peticiones del cliente, como mecanismos de soporte en la construcción y mantenimiento del *conocimiento*

compartido; y (2) explotar el *conocimiento compartido* construido como consecuencia del punto anterior, con objeto de alcanzar un entendimiento profundo del escenario grupal para finalmente distribuirlo a todos los miembros del grupo a través de la IU, tratando de contribuir y promover una colaboración real. Estos mecanismos se materializan en el FPE-C a través de las *reglas de colaboración* descritas anteriormente y las *directrices de colaboración*, que se conjugan con aquéllas para participar en la toma de decisiones relativas al grupo, identificar las necesidades de interacción entre los distintos integrantes y asistir y reforzar las acciones a tomar en favor de una mayor comunicación, coordinación y colaboración. Estas últimas son las tres metas principales del campo de *groupware*, identificadas por Ellis et al. en [EGR91] como las tres áreas claves para soportar una correcta interacción de grupo. Según estos autores, una colaboración efectiva requiere compartir información entre los miembros participantes. Estos mecanismos persiguen alcanzar ese *conocimiento compartido* como aspecto esencial en entornos multi-usuario.

La intervención de las *directrices de colaboración* en el proceso queda plasmada en la figura 4.2 mediante la conexión de esta componente con el *repositorio de modelos 1* y *2*, con objeto de intervenir, al igual que las *reglas de colaboración* en las tres primeras fases de la derivación de la IU, o bien en la fase de *Proceso de Inferencia Abstracto* y *Preparación de los modelos iniciales*, si se aplica *abstracción*.

Es evidente que este escenario de explotación de la situación de grupo no sería posible sin que en el lado del cliente se hiciera lo posible por capturar cada una de las percepciones individuales (*consciencia de grupo particular*), poniendo en funcionamiento los oportunos *mecanismos locales de consciencia de grupo* (*Definición 2.2*; *Capítulo 2*), aspecto del que se ocupa, entre otras cosas, el *Motor de Plasticidad Implícita* que se describe en el *Capítulo 6*.

4.1.4. Proceso de derivación de IUs

El proceso de generación de la IU propuesto en este marco conceptual pretende brindar la máxima flexibilidad. En efecto, cuanto mayor flexibilidad, mayor número de herramientas podrán ser representadas. Precisamente, éste es el objetivo perseguido con su definición. No se establece, por tanto, un proceso secuencial rígido, marcado por la sucesión de las diferentes fases en que se estructura el proceso. Existen una serie de factores que contribuyen a esta flexibilidad, los cuales se enumeran a continuación.

1. la posibilidad de aplicar operaciones de *adaptación*, las cuales permiten reconfigurar la *IU Abstracta* y la *IU Concreta*, así como también los modelos del primer nivel de

abstracción, si fuera necesario, sin tener que recurrir a un nivel previo para adecuar un determinado diseño a una nueva situación contextual.

2. la posibilidad de activar el proceso en cualquiera de sus fases. Tal y como se refleja en la figura 4.2 (pág. siguiente), si se distingue entre las diversas modalidades de obtención de la IU, pueden identificarse hasta un total de siete *puntos de entrada*[5] en el proceso de construcción de IUs. Éstos se representan a través de flechas dobles flotantes en la parte izquierda de cada una de las herramientas de desarrollo automático.

3. la posibilidad de intervención del usuario, tanto en la construcción de los modelos iniciales como en las fases intermedias, con objeto de refinar diversos aspectos de la IU, así como de velar por el cumplimiento de los criterios de usabilidad.

4. la posibilidad de combinar operaciones de *reificación* (representada mediante flechas de color negro en sentido descendente) con operaciones de *abstracción* (representada mediante flechas de color negro en sentido ascendente), ambas definidas a continuación, haciendo factible cualquier trayectoria posible en la obtención de la IU.

La conjunción de todos estos factores hace factible cualquier modalidad de explotación de los modelos, otorgando absoluta libertad para intercalar operaciones de *reificación*, *abstracción* y *adaptación*, y permitiendo por tanto avanzar, retroceder, acomodar, refinar y activar el proceso según convenga. La figura siguiente muestra el marco conceptual propuesto.

4.1.4.1. Operaciones soportadas

Tal y como se expone arriba, el proceso es definido como una combinación de tres operaciones: *reificación*, *abstracción* y *adaptación*. Además, interviene una cuarta operación denominada *introspección*, que es la operación encargada de actualizar en el *servidor de plasticidad* el conocimiento disponible sobre la IU en uso, en el momento de la activación del *Motor de Plasticidad Explícita*, tras haber examinado el estado de la IU en la plataforma cliente. A continuación se describen cada una de ellas.

[5]Indicador del nivel de reificación en el cual puede empezar el proceso de desarrollo, tal y como se entienden en el marco de referencia unificado (véase *sección 3.3.4*).

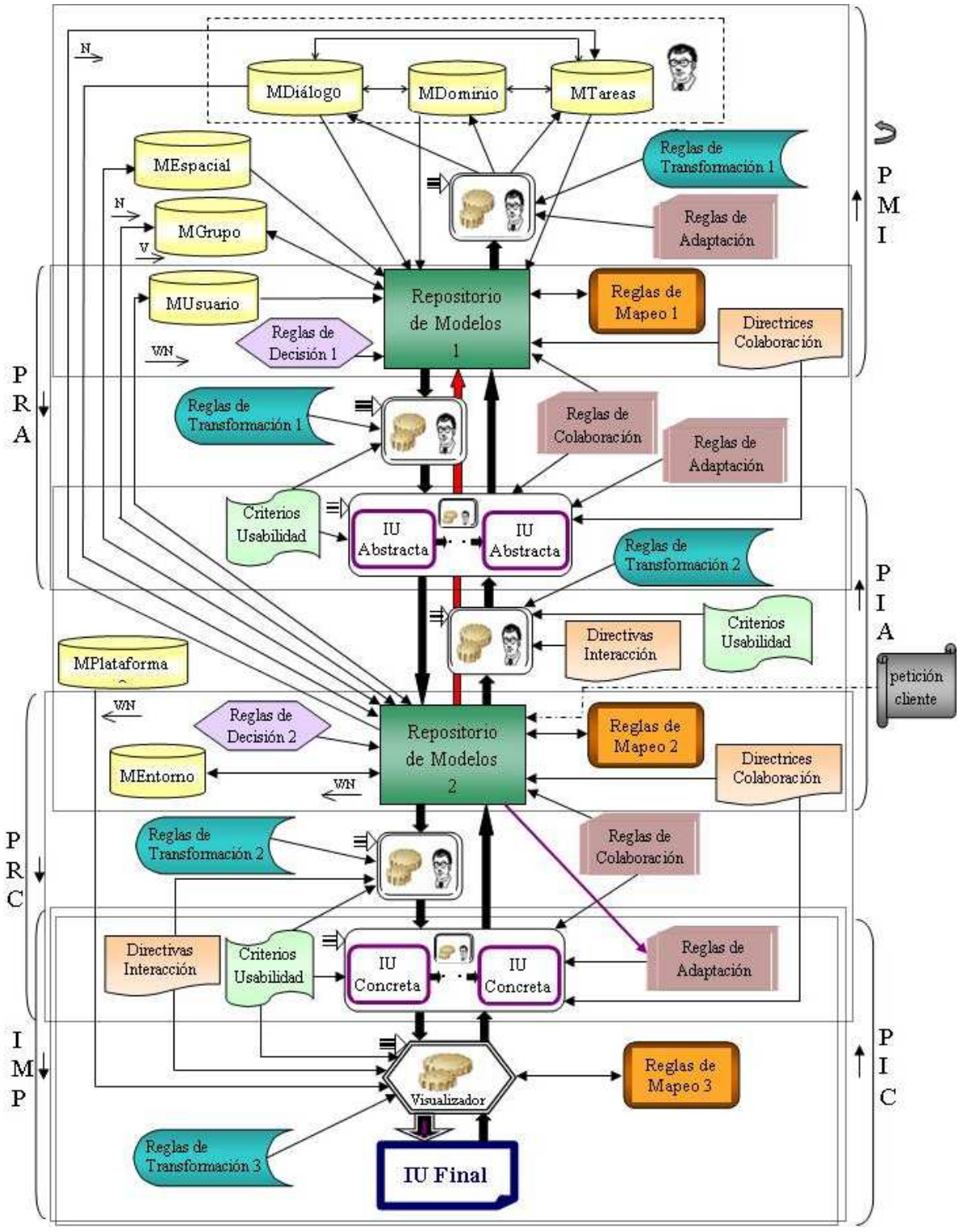


Figura 4.2: El Marco Conceptual propuesto denominado *Framework de Plasticidad Explícita Colaborativo.*

4.1.4.1.1. Reificación (derivación vertical descendente)

Operación de transformación de un modelo para un contexto de uso dado a un modelo más concreto para el mismo contexto de uso. Un proceso de reificación puro cubre el proceso de derivación de los sucesivos *modelos transitorios*, partiendo del primer nivel de abstracción hasta llegar al nivel operativo (*IU Final*), como resultado de aplicar sucesivos pasos de refinamiento hacia la obtención de descripciones lógicas de la IU cada vez más concretas y específicas para el contexto de uso al que vaya a destinarse.

Esta operación se representa en la figura 4.2 a través de las flechas verticales de color negro en sentido descendente, que indican el camino de derivación de la *IU Final*.

4.1.4.1.2. Abstracción (inferencia vertical ascendente)

Operación que complementa el proceso de *reificación* de manera que permite estructurar el desarrollo de IUs mediante la inferencia de diseños abstractos de la IU a partir de diseños más concretos, es decir, en *bottom-up*. Así por ejemplo, un prototipo de *IU Abstracta* puede ser inferido a partir de una *IU Concreta*, los *modelos del nivel superior de abstracción* pueden ser recuperados a partir de una *IU Abstracta*, e incluso la obtención de diseños abstractos a partir de *IUs Finales* en código fuente. Se trata de seguir un proceso de *ingeniería inversa* en la producción de IUs [Bou06]. Se define la operación de *abstracción* como la operación que transforma una representación de la IU desde un nivel no inicial de abstracción a otro nivel de abstracción superior.

Es evidente que esta operación otorga considerable flexibilidad al proceso, así como una mayor reutilización de los modelos y diseños, al ofrecer una alternativa y/o un complemento al proceso de reificación puro. La posibilidad de traducir una *IU Final* a otra para otro contexto de uso pasando, si es necesario, por diseños abstractos, obtenidos desde el nivel operativo, revoluciona la concepción tradicional acerca del proceso de generación de IUs. Si la posibilidad de elegir libremente el punto de entrada introduce notable flexibilidad (en la práctica el modelado de tareas suele obviarse), la incorporación de la operación de *abstracción* posibilita cualquier tipo de instanciación del marco conceptual en cualquier tipo de herramienta, ofreciendo absoluta libertad en la disponibilidad de medios y recursos para no tener que empezar siempre desde cero en el proceso.

Si a esto se añade la posibilidad de utilizar las *reglas de mapeo* para facilitar la inferencia de modelos abstractos a partir de otros más concretos, la flexibilidad es aún mayor, así como también la reutilización de diseños y, muy importante, decisiones acerca de procesos previos.

Aunque son pocas las herramientas basadas en modelos construidas incorporando la ingeniería inversa, sus beneficios son considerables, sobre todo de cara a su aplicación práctica. Todo esto induce a pensar que sea factible su proliferación en un futuro próximo. Las herramientas más conocidas que aplican ingeniería inversa son las que se mencionan en el *Capítulo 3* (véase *sección 3.5.*): Vaquita [BVS02] –la tesis de Laurent Bouillon [Bou06] está destinada a avanzar en este tema dentro del marco de trabajo del proyecto CAMELEON-, su versión *on-line* ReversiXML[6] [BVC04], que transforma una página Web en HTML a UsiXML, tanto al nivel de *IU Abstracta* como al de *IU Concreta*, así como a una página Web para otra plataforma distinta. Otro ejemplo es WebRevEnge [PP02], todas ellas presentadas en el *Capítulo 3* (véase *sección 3.5*).

Una posibilidad para materializar la incorporación de la ingeniería inversa en el proceso de desarrollo de una herramienta basada en modelos concreta es la de sustentar todo el proceso a través del lenguaje usiXML. UsiXML es un lenguaje de descripción de la IU (véase *Capítulo 3*; *sección 3.1.3.*) diseñado para soportar un desarrollo multi-direccional de la IU, permitiendo su especificación en múltiples niveles de abstracción conforme al camino de desarrollo seguido (ascendente o descendente), y en múltiples niveles de independencia (plataforma, modalidad de interacción, contexto de uso, etc.) [LVM$^+$04]. Eso significa que una IU puede ser especificada y producida a y desde diferentes y múltiples niveles de abstracción, manteniendo el mapeo entre estos niveles. Así, el proceso de desarrollo puede iniciarse en cualquier nivel de abstracción, obtener una o varias *IU Finales* para diversos contextos de uso (*forward engineering*), recuperar la *IU Final* a cualquier nivel superior de abstracción (*reverse engineering*), y por último, adaptar la IU a cualquier nivel de abstracción (*reengineering*) [LV04].

Esta flexibilidad se corresponde con lo que se puede denominar el paradigma de desarrollo de IUs multi-direccional (*multi-path UI development*) [LVM$^+$04]. Las técnicas subyacentes que formalizan usiXML son técnicas de transformación de grafos. De hecho, la combinación de transformaciones establece distintos caminos de desarrollo. Con objeto de afrontar el desarrollo de IUs multi-direccional de manera genérica, usiXML está equipado con una colección de modelos de IU básicos, entre los que se encuentra el denominado *modelo de transformación* [LVM$^+$04], como modelo que describe las transformaciones entre modelos.

La operación de abstracción se representa en la figura 4.2 a través de las flechas verticales de color negro en sentido ascendente que indican el camino de recuperación de diseños cada vez más abstractos de la IU.

[6] http://www.usixml.org/index.php?view=page&idpage=17&screen_size=1024x768

4.1.4.1.3. Adaptación horizontal

Operación de transformación de un *modelo transitorio* (nivel abstracto y concreto) o de un modelo del primer nivel de abstracción para un contexto dado al mismo tipo de modelo para otro contexto de uso, como operación que soporta la adaptación a los *cambios contextuales* (*Definición 2.4*; *sección 2.1.3*) a través de la aplicación de las *reglas de adaptación*. La obtención de una nueva *IU Final* para una nueva plataforma o lenguaje de programación debe pasar necesariamente por la fase de *Implementación*, donde un *visualizador* específico transforma la *IU Concreta* en otra versión en código fuente. Eso significa que una *IU Final* no puede ser objeto de adaptación.

Esta operación aplicada a los *modelos transitorios* evita tener que reiniciar el paso de reificación/abstracción en los niveles de abstracción superiores/inferiores para obtener nuevas versiones de la IU, adecuadas a la nueva situación contextual, valiéndose de versiones ya existentes. Esto constituye otro ejemplo de reutilización, en este caso de diseños previos de la IU, que un enfoque basado en modelos hace posible. Se representa a través de una derivación horizontal de una *IU Abstracta* o *IU Concreta* partiendo de una *IU Abstracta* o *IU Concreta* previa en los niveles abstracto y concreto.

La operación de *adaptación* aplicada a los modelos situados en el primer nivel de abstracción permite que sean adecuados a una nueva situación contextual (nueva plataforma, entorno, usuario, o incluso situación grupal). Como ya se ha mencionado, el ejemplo de adaptación más frecuente en este nivel es el de ejercer una poda o una expansión en el *modelo de tareas*. Estas operaciones se producen únicamente ante la notificación de aquellos cambios contextuales que tienen una mayor repercusión, como por ejemplo, pasar a utilizar un dispositivo de características significativamente distintas (perteneciente a otra familia de dispositivos), un cambio sustancial en la situación relativa al entorno, cambios significativos en las preferencias o necesidades del usuario, o bien un cambio relevante en la situación relativa al trabajo en grupo. En estos casos, esta operación de adaptación se representa por medio de una flecha de retorno hacia los *modelos del nivel superior de abstracción*, partiendo de las herramientas automáticas de la primera fase.

La figura 4.3 (pág. siguiente) muestra los puntos en los que puede llevarse a cabo una operación de *adaptación* a lo largo del proceso, juntamente con los componentes que intervienen.

En definitiva, la combinación de las tres operaciones (*reificación*, *abstracción* y *adaptación*) evita repetir todo el proceso, permitiendo la reutilización de modelos intermedios o superiores, de cara a la obtención de diversas versiones de la IU, permitiendo materializar el *modelo de la interfaz* de diversas maneras.

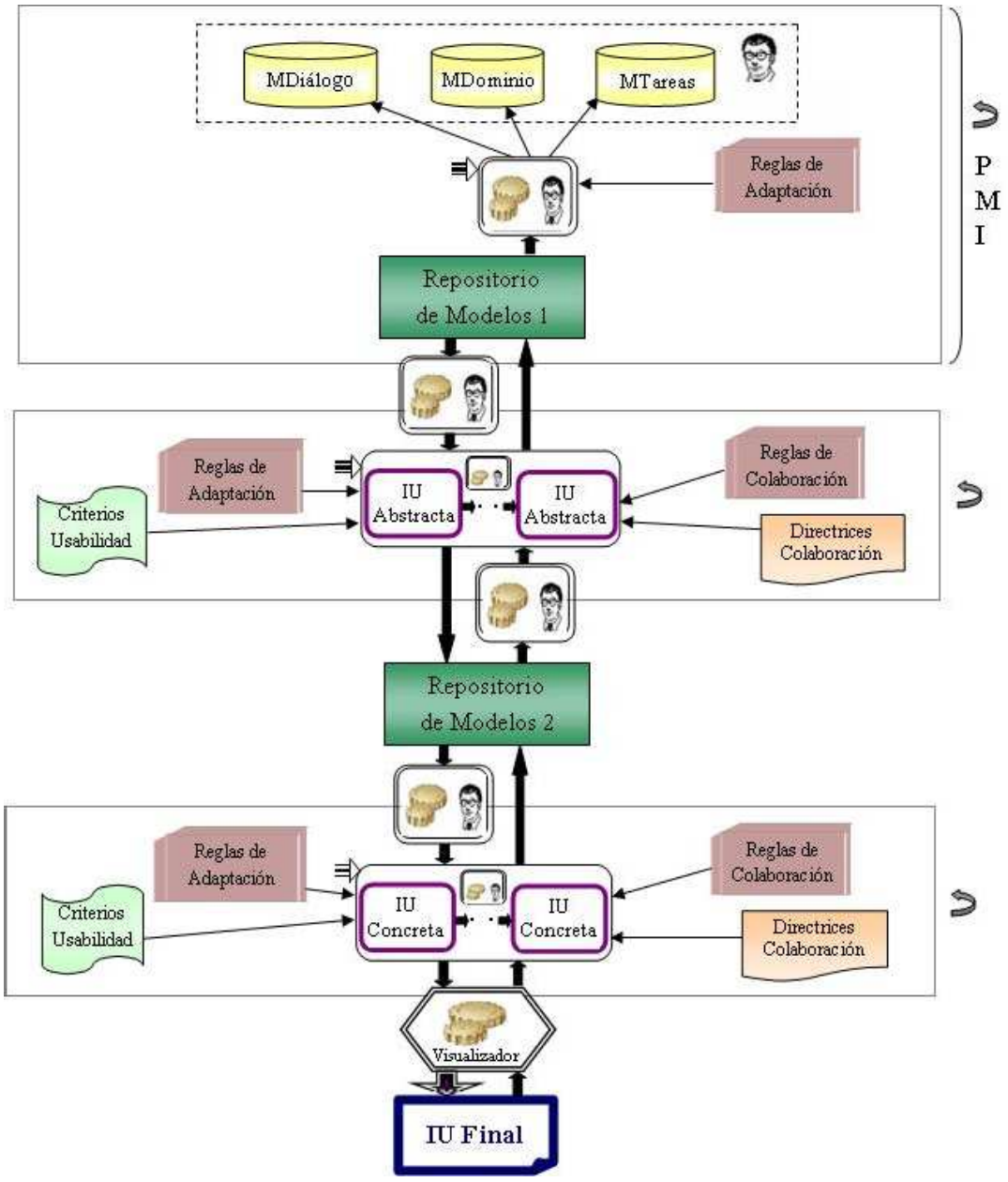


Figura 4.3: Proceso de adaptación y componentes involucradas.

4.1.4.1.4. Introspección

Operación de actualización de los *modelos contextuales* cada vez que se recibe una petición explícita por parte de la plataforma cliente, como consecuencia de un *cambio contextual* (*Definición 2.4*; *sección 2.1.3*), tras un periodo de utilización del sistema. Es evidente que bajo el enfoque basado en la *visión dicotómica*, el *servidor de plasticidad* requiere ponerse al día acerca de la situación en curso, antes de proceder al proceso de construcción de la IU. No hay que descuidar que el objetivo perseguido es el de generar una IU lo más ajustada posible a la nueva situación, comunicada por parte de la plataforma cliente. Se trata por tanto de la operación que hace efectivo el último eslabón de la cadena en la propagación de los *cambios contextuales* producidos durante el uso de la IU, haciendo llegar esos cambios, recogidos temporalmente en el *repositorio de modelos 2*, a los *modelos contextuales* subyacentes. Tal y como se expone la *sección 4.1.5.1*, esta operación tiene dos posibles modalidades: *introspección por notificación* e *introspección por variación*.

Se utiliza el término *'introspección'*, perteneciente al campo de la *computación reflexiva* [Mae87], [Smi84], ya que todo este mecanismo de realimentación entre ambos lados de la arquitectura cliente-servidor puede asimilarse en cierto modo a una *arquitectura* o *sistema reflexivo*[7]. En efecto, según [Zim96], bajo una arquitectura reflexiva, un sistema se considera compuesto de dos partes: la parte de la aplicación (*nivel base*), que corresponde al sistema interactivo en uso en el lado del cliente, y una parte reflexiva (*nivel meta*) capaz de razonar y actuar sobre sí misma, que en este caso corresponde al *modelo de interfaz* almacenado en el *servidor de plasticidad*.

El *nivel meta* proporciona una *auto-representación*[8] del propio sistema, que en este caso consiste en la descripción de la IU recogida en el conjunto de modelos declarativos (su *modelo de interfaz*), entre ellos los que reúnen las propiedades definidas por el entorno de ejecución. Además, en un sistema reflexivo, esta representación es susceptible de *introspección* y adaptación, y además está *causalmente conectada*[9] al comportamiento subyacente. Ya se ha visto que el *modelo de interfaz* del sistema interactivo cumple las propiedades de adaptación y de estar *causalmente conectado*. En particular, la capacidad de *introspección* consiste en la posibilidad de examinar el estado del sistema subyacente en un momento dado –en este caso su IU-, haciendo posible que los cambios producidos en

[7]Sistema que es capaz de razonar y mantener información sobre sí mismo. Más formalmente se define como un sistema capaz de manipular, alterar, inspeccionar y mantener una auto-representación de sí mismo, con objeto de extender y adaptar su propia computación.

[8]Modelado o representación explícita de su propio comportamiento, operaciones, estructura interna, y cuantas propiedades no funcionales en las que se esté interesado, las cuales son cambiantes.

[9]Significa que los cambios producidos en la auto-representación son inmediatamente reflejados en el estado y comportamiento del sistema subyacente (propiedad de reflexión), y vice-versa (propiedad de introspección).

el mismo durante su uso sean transmitidos a su *auto-representación*. Este es precisamente el propósito de la operación de actualización de los *modelos contextuales*, y por ese motivo se ha convenido denominarla operación de *introspección*.

La operación de *introspección* se representa a través de las flechas etiquetadas que parten del *repositorio de modelos 2* hacia cada uno de los *modelos contextuales* y también hacia el *modelo de tareas*, tal y como se detalla más adelante.

4.1.4.2. Intervención del humano

En cuanto a la intervención del experto humano, se fomenta una iniciativa mixta que promueve una operación semi-automática en la que se puede combinar la acción manual por parte del experto humano con la acción automática a partir de las especificaciones [Hor99]. De hecho, la generación automática de IUs no ha tenido una amplia aceptación en el pasado [MHP00]. Así, las herramientas automáticas infieren descripciones que a continuación el diseñador tiene la posibilidad de ajustar de acuerdo a los requisitos específicos de usabilidad o de calidad que sean necesarios. Por ejemplo, es recomendable proporcionar herramientas de edición que permitan al desarrollador alterar, refinar y personalizar la apariencia de los distintos *modelos de presentación*. Se promueve esta combinación tanto en las operaciones de *reificación* como en las de *abstracción* y *adaptación*.

En síntesis, la clave de esta iniciativa mixta es que el experto humano pueda intervenir en todos los pasos, si así se considera oportuno, con objeto de involucrarlo en todo el ciclo de vida de desarrollo de la IU. Precisamente, el cumplimiento de este requisito hace que las herramientas basadas en modelos se puedan encuadrar dentro de los métodos *centrados en el usuario* [Nor86].

4.1.5. Fases que componen el Marco Conceptual

Puesto que el marco conceptual propuesto contempla un paradigma de desarrollo de IUs multi-direccional (*multi-path UI development*) [LVM$^+$04] y, en consecuencia, los sucesivos diseños de la IU pueden obtenerse por *reificación* o por *abstracción* (aplicación de la ingeniería inversa), es natural que la concepción de algunas fases cambie en función de si se está aplicando un proceso de *reificación* o de *abstracción*, particularmente en el caso de los niveles abstracto y concreto. Además, también el conjunto de componentes a intervenir puede variar. Es por ello que distinguimos hasta un total de seis fases: cuatro correspondientes a la aplicación de un proceso de derivación, y dos específicas a la aplicación de un proceso de abstracción. La fase de *Preparación de los Modelos Iniciales* es

común en ambos procesos. Con objeto de introducir todas estas fases, se exponen a continuación distinguiéndolas según el camino seguido (*top-down o derivación* y *bottom-up o inferencia*).

4.1.5.1. Proceso de derivación descendente

La sucesión de fases en la derivación de la IU siguiendo un proceso de reificación puro es la que aparece reflejada en la figura 4.4 y se describe en detalle a continuación.

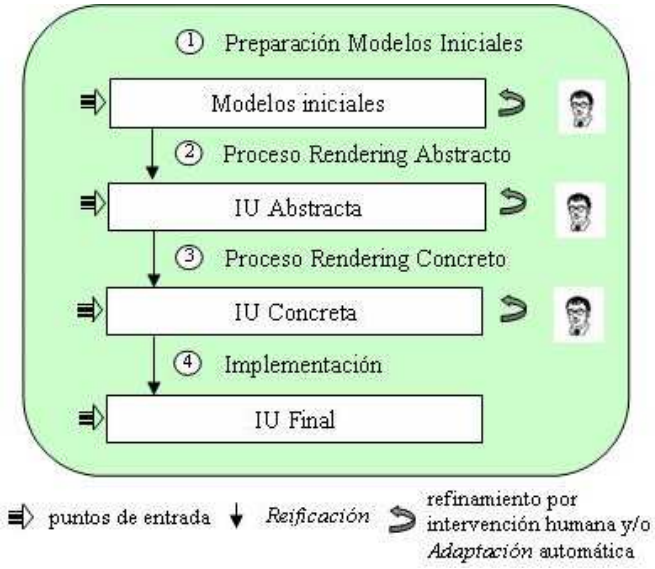


Figura 4.4: Sucesión de fases en el proceso de derivación descendente.

4.1.5.1.1. Preparación de los modelos iniciales

Esta fase (*PMI* en la figura 4.2) hace referencia a la preparación de los *modelos iniciales*, concretamente de los correspondientes al nivel superior de abstracción (*modelo de tareas*, de *dominio* y de *diálogo*) para la nueva generación de una IU, puesto que son los que (1) son específicos de cada sistema interactivo; y (2) pueden ser objeto de adaptación (aplicación de la operación de *adaptación*, presentada anteriormente, utilizando las *reglas de adaptación*), con objeto de ser acomodados a una nueva situación contextual o situación grupal Las restricciones contextuales recibidas de la plataforma cliente son reunidas en el *repositorio de modelos 1* en estos casos (conexión *repositorio de modelos 2* con *repositorio de modelos 1* a través de la flecha ascendente de color rojo). Así pues, como estos modelos no tienen por qué permanecer invariables a lo largo de las sucesivas activaciones del *Motor de Plasticidad Explícita*, y por consiguiente este paso de adaptación constituye en ocasiones un paso más

en la generación de una IU determinada. Es por ello que esta fase es considerada como parte integral del proceso de producción de IUs descrito a través del marco conceptual.

Esta fase representa tres situaciones distintas: (1) la especificación de los modelos inicialmente, como primera versión que entra en juego en la derivación de la IU; (2) sus posteriores adaptaciones con objeto de ser adecuados a una nueva situación contextual, como consecuencia de las sucesivas activaciones del *Motor de Plasticidad Explícita* (de ahí que aparezcan en el gráfico -véase figura 4.5- las herramientas automáticas con el correspondiente *punto de entrada* también en esta fase); y (3) la posibilidad de inferir estos modelos a partir de una versión previa de la *IU Abstracta*, aplicando ingeniería inversa (sentido ascendente de las flechas), donde intervienen las *reglas de transformación* 1 y se registran las correspondencias entre elementos de ambos niveles en las *reglas de mapeo 1*. No es necesario en este caso distinguir dos fases distintas para diferenciar un paso de derivación o de inferencia.

En concreto, la especificación inicial de los modelos suele estar soportada por herramientas de modelado, generalmente gráficas, que facilitan su construcción.

La figura 4.5 recoge los componentes de esta fase. Como ya se ha cometado, las *reglas* y *directivas de colaboración* también intervienen en este nivel.

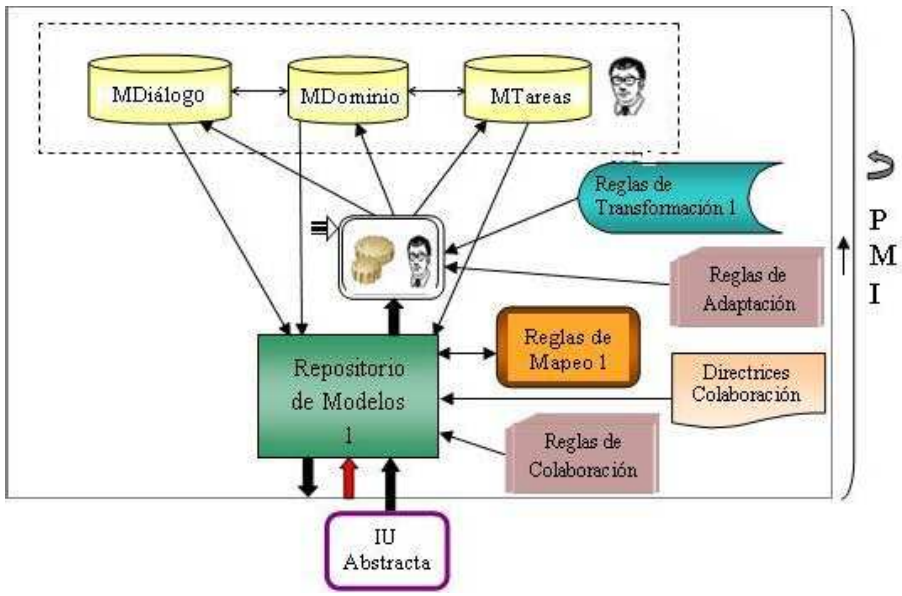


Figura 4.5: Fase *Preparación de Modelos Iniciales*.

4.1.5.1.2. Proceso de Rendering Abstracto

Esta fase (*PRA* en la figura 4.2) tiene como objetivo derivar la *IU Abstracta* -o *modelo de presentación* abstracto- a partir de una descripción suficientemente detallada, tanto de las tareas de interacción como de los conceptos de dominio, como de la especificación de las transiciones entre estados de la IU, es decir, a partir de los *modelos iniciales* pertenecientes al primer nivel de abstracción (*modelo de tareas*, *modelo de diálogo* y *modelo de dominio*), mediante la transformación descrita por las *reglas de transformación 1*. Estas relaciones entre modelos pueden ser especificadas a través de las *reglas de mapeo 1*.

No obstante, algunos de los *modelos contextuales* también comienzan a ejercer su influencia a este nivel de abstracción, entrando en consideración desde un principio las dependencias y consecuentes referencias existentes entre éstos y los aspectos relativos al primer nivel de abstracción (tareas, objetos de dominio y transiciones entre estados de la IU). Se trata del *modelo de usuario* -con objeto de ir personalizando la IU al perfil de usuario en cuestión-, el *modelo espacial* –gran parte de las tareas pueden estar supeditadas a la localización geográfica donde se encuentre ubicado el usuario- y el *modelo de grupo* -algunas situaciones relativas al grupo de trabajo pueden ser determinantes en el uso que parte de los integrantes del grupo deba hacer del sistema en un momento dado, esto es, en las tareas a plasmar en la IU. De hecho, ya se ha expuesto que la relación entre el *modelo de tareas* y el *modelo de grupo*, así como entre el *modelo de diálogo* y *el modelo de grupo* cobran una importancia primordial en entornos colaborativos (*Recomendación 1*, *Heurística 4.6* y *Heurística 4.4*, respectivamente).

En definitiva, se trata de generar parcialmente la IU a partir de la estructura del *modelo de tareas* y de definir la navegación entre ventanas y cuadros de diálogo, valiéndose de un conjunto de *reglas de transformación* (las *reglas de transformación 1* en combinación con conceptos capturados del *modelo de usuari*o, *modelo espacial* y *modelo de grupo*, y donde también intervienen los *criterios de usabilidad* y por último las *reglas de colaboración* y las *directrices de colaboración*, si se trata de un sistema colaborativo.

Es recomendable que el *modelo de presentación* abstracto se almacene en un formato basado en XML. De ese modo el diseñador puede crear nuevas reglas de transformación usando cualquiera de los procedimientos habituales aplicables a la transformación de una especificación XML, como son transformaciones XSLT[10], transformaciones de gramáticas de grafos [Lim04] o transformación de especificaciones algebraicas [BCR05]. La figura 4.6 (pág. siguiente) recoge los componentes de esta fase.

4.1.5.1.3. Proceso de Rendering Concreto

[10] http://www.w3.org/TR/xslt

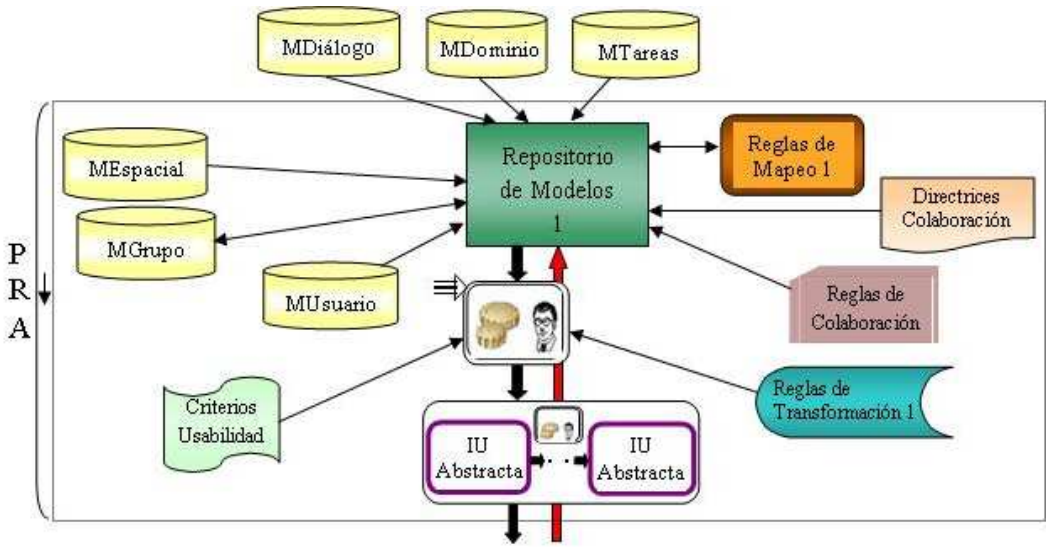


Figura 4.6: Fase *Proceso de Rendering Abstracto*.

Esta fase (*PRC* en la figura 4.2) se encarga de obtener una *IU Concreta* a partir de la *IU Abstracta* obtenida en la fase anterior, mediante la transformación descrita a través de las *reglas de transformación 2*.

La *IU Concreta* representa la IU en un nivel de abstracción bastante cercano al código final, y por tanto debe ser capaz de especificar con detalle la presentación que el usuario verá finalmente. Esta descripción más refinada resulta de la selección y composición del conjunto de *objetos concretos de interacción* –definidos en el *Capítulo 3* (véase *sección 3.2.2.*)- de acuerdo a la información contextual representada en los *modelos contextuales* (*modelo de usuario*, *modelo espacial*, *modelo de plataforma*, *modelo de entorno* y *modelo de grupo*), y gobernado por el *modelo de diálogo*, tal y como se refleja en la figura 4.7 (pág. siguiente).

Más concretamente, esta fase consiste en decidir qué componente o conjunto de componentes concretas de la IU representará la funcionalidad descrita por cada componente o conjunto de componentes abstractas, decisiones en las intervienen las *reglas de transformación 2* y que, opcionalmente, pueden ser recogidas en las *reglas de mapeo 2*, con objeto de mantener la trazabilidad del proceso. Por ejemplo, de una *IU Abstracta* se podría derivar una IU textual, gráfica o auditiva (también conocida como vocal). El decantarse por una u otra opción se hará en función de la información contextual (preferencias del usuario, nivel de ruido ambiental, plataforma de interacción o situación grupal). De este modo se lleva a cabo el proceso de refinamiento hacia la obtención de la IU apropiada a cada circunstancia contextual y a las restricciones del grupo, si es el caso. En definitiva, un mismo *objeto abstracto de interacción*, con las mismas propiedades, puede ser representado utilizando

distintas combinaciones de *objetos concretos de interacción*. Por consiguiente, este proceso de transformación de la *IU Abstracta* en la *IU Concreta* consiste, en definitiva, en un proceso de propagación de las restricciones impuestas por las circunstancias contextuales reflejadas en los *modelos contextuales*. En general, la selección de *elementos de interfaz* para representar cada *objeto abstracto de interacción* no responde a una correspondencia uno a uno.

Para que el marco conceptual pueda considerarse apropiado para el desarrollo de IUs plásticas, es imprescindible que este proceso de transformación esté orientado hacia la maximización de la usabilidad del sistema. Es por ello que en esta fase intervienen los *criterios de usabilidad* y las *directivas de interacción*. Otras componentes que también intervienen en esta fase son las *reglas de colaboración* y las *directrices de colaboración*, en el caso de tratarse de un sistema colaborativo.

Por último, es importante remarcar que la transformación de la *IU Abstracta* en *IU Concreta* debe poderse realizar tanto de manera automática como semi-automática, es decir, siguiendo una iniciativa mixta. Las propuestas de transformación totalmente automática, como son los árboles de selección [Van99], [BHLV94] o las reglas de selección [Van97], [JWZ93], no obtienen unos resultados de calidad aceptables desde el punto de vista de la usabilidad. Eso ha dado lugar a que la mayoría de las aproximaciones propongan hoy en día un proceso de transformación semi-automático, es decir, siguiendo una iniciativa mixta. En particular, la selección del modo en que los elementos de la interfaz se distribuyen dentro de los contenedores es recomendable que sea refinada manualmente, con objeto de mejorar la usabilidad de la IU creada, a partir de la cual partirá el proceso de adaptación dinámica en la plataforma cliente. La figura 4.7 recoge los componentes de esta fase.

4.1.5.1.4. Implementación

Esta tercera fase (*IMP* en la figura 4.2) genera una *IU Final* a partir de la *IU Concreta* obtenida en la fase previa. En este marco conceptual se propone su automatización a través de los llamados *visualizadores* (o también *renderers*), capaces de transformar el aspecto visual y comportamiento dinámico de la *IU Concreta* -generalmente especificada en un lenguaje intermedio basado en XML- en código directamente compilable o interpretable.

Las otras opciones existentes, consistentes en obtener una especificación de la IU desarrollada, adecuada para un entorno de desarrollo de IUs determinado, como es el caso de la herramienta TADEUS [ES95], no son tan versátiles e independientes del entorno de ejecución al que se llevará el sistema. Cabe señalar que, en contrapartida, en el primer caso

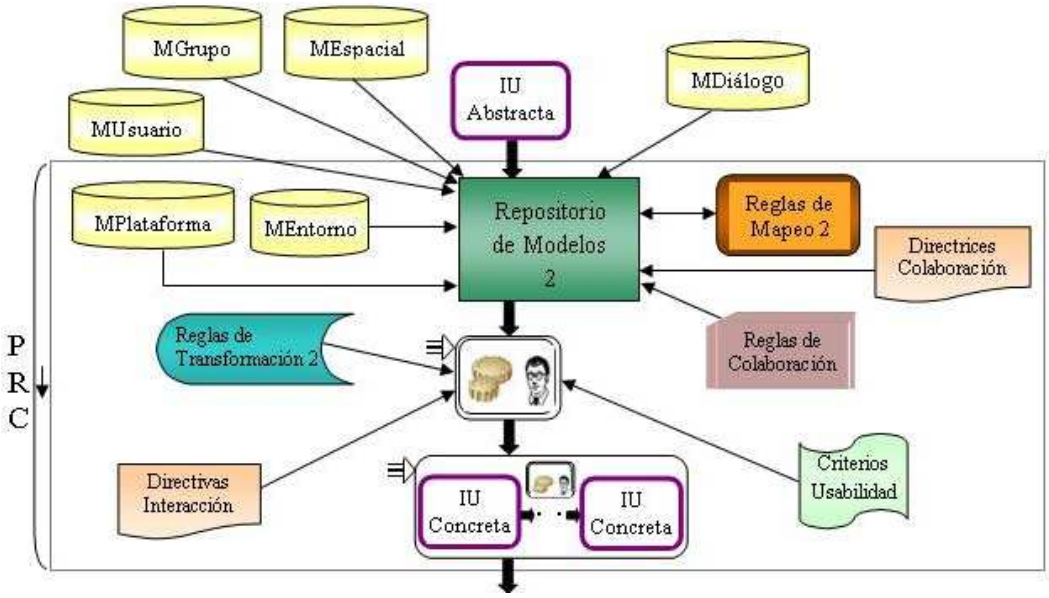


Figura 4.7: Fase *Proceso de Rendering Concreto*.

se requiere la construcción de un *visualizador* diferente para cada lenguaje o plataforma prevista. El marco conceptual permite la representación de ambas opciones.

Los componentes que intervienen en este paso son las *reglas de transformación 3* (en ocasiones integradas en los *visualizadores* de manera implícita), los *criterios de usabilidad* y *directivas de interacción*, así como, por supuesto, las especificaciones de la plataforma objetivo descritas a través del *modelo de plataforma*. Las *reglas de mapeo 3*, en el caso de incluirse, recogerían las relaciones establecidas entre ambos *modelos de presentación*. La figura 4.8 recoge los componentes de esta fase.

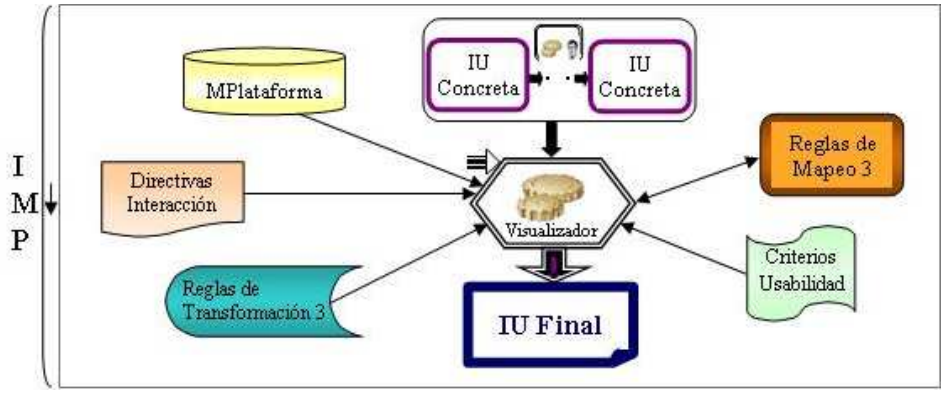


Figura 4.8: Fase *Implementación*.

4.1.5.2. Proceso de inferencia ascendente

La sucesión de fases en la inferencia de los diseños abstractos a partir de una IU ejecutable, siguiendo un proceso de abstracción puro aparece reflejada en la figura 4.9.

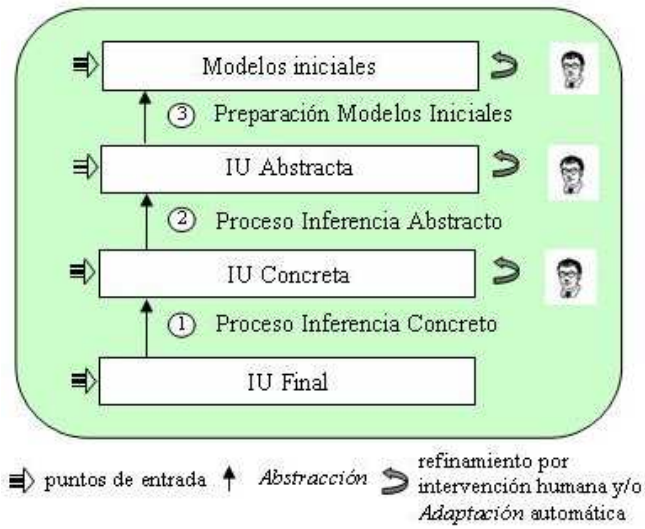


Figura 4.9: Sucesión de fases en el proceso de inferencia ascendente.

4.1.5.2.1. Proceso de Inferencia Concreto

Esta fase (*PIC* en la figura 4.2) se encarga de obtener una *IU Concreta* –o modelo de presentación concreto- a partir de una *IU Final* que puede haber sido obtenida anteriormente, o bien proporcionada expresamente como *input* al proceso, con objeto de proceder a su transformación para ser acomodada a otro contexto de uso, aplicando ingeniería inversa.

La descripción al nivel concreto se obtendrá de la inferencia del conjunto de *objetos concretos de interacción* que se corresponden con los *widgets* finales, de acuerdo a las *reglas de transformación 3*. No obstante, si tal y como se ha mencionado arriba, esa *IU Final* ya se había obtenido con anterioridad, se puede recurrir a la información de trazabilidad –registro de las relaciones entre *modelos de presentación* aplicada en un proceso de derivación previo- registrada a través de las *reglas de mapeo 3*.

En esta fase también intervienen el *modelo de plataforma*, las *directivas de interacción* y los *criterios de usabilidad*.

La figura 4.10 (pág. siguiente) recoge los componentes que intervienen en esta fase.

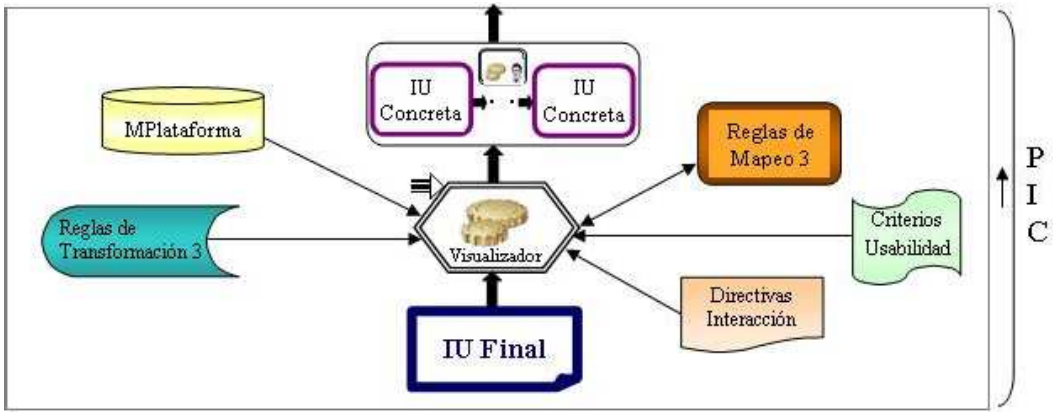


Figura 4.10: Fase *Proceso de Inferencia Concreto*.

4.1.5.2.2. Proceso de Inferencia Abstracto

Esta fase (*PIA* en la figura 4.1) tiene como objetivo derivar la *IU Abstracta* -o *modelo de presentación* abstracto- a partir de una *IU Concreta*, a transformar aplicando ingeniería inversa.

Se trata de inferir una representación abstracta de la IU a partir de una representación más concreta, valiéndose de las *reglas de transformación 2*, en combinación con las *reglas de colaboración* y *directrices de colaboración*, si es el caso, los *criterios de usabilidad*, *directivas de interacción*, así como los conceptos capturados de los *modelos contextuales*, que alimentan el *repositorio de modelos 2*. En esta fase puede ser útil recurrir a las *reglas de mapeo 2*, en el caso de registrarse la información de trazabilidad correspondiente a procesos previos de derivación o inferencia.

La figura 4.11 (pág. siguiente) recoge los componentes que intervienen en esta fase.

4.1.5.2.3. Preparación de los modelos iniciales

Por último, la fase de *Preparación de los modelos iniciales*, en su modalidad de abstracción pura sería el último paso en este proceso de ingeniería inversa hacia la inferencia de los modelos abstractos, fase que coincide con la primera a llevar a cabo en un proceso de derivación puro, ya introducida. La figura 4.5 anterior refleja esta fase tanto habiendo seguido un proceso de abstracción como de derivación. De hecho, la conexión de la *IU Abstracta* con el *repositorio de modelos 1* a través de una flecha ascendente corresponde a un paso específicamente de abstracción. Igual que en la fase previa, en esta fase puede resultar útil recurrir a las *reglas de mapeo 1* para deducir la inferencia de los *modelos del nivel superior de abstracción*.

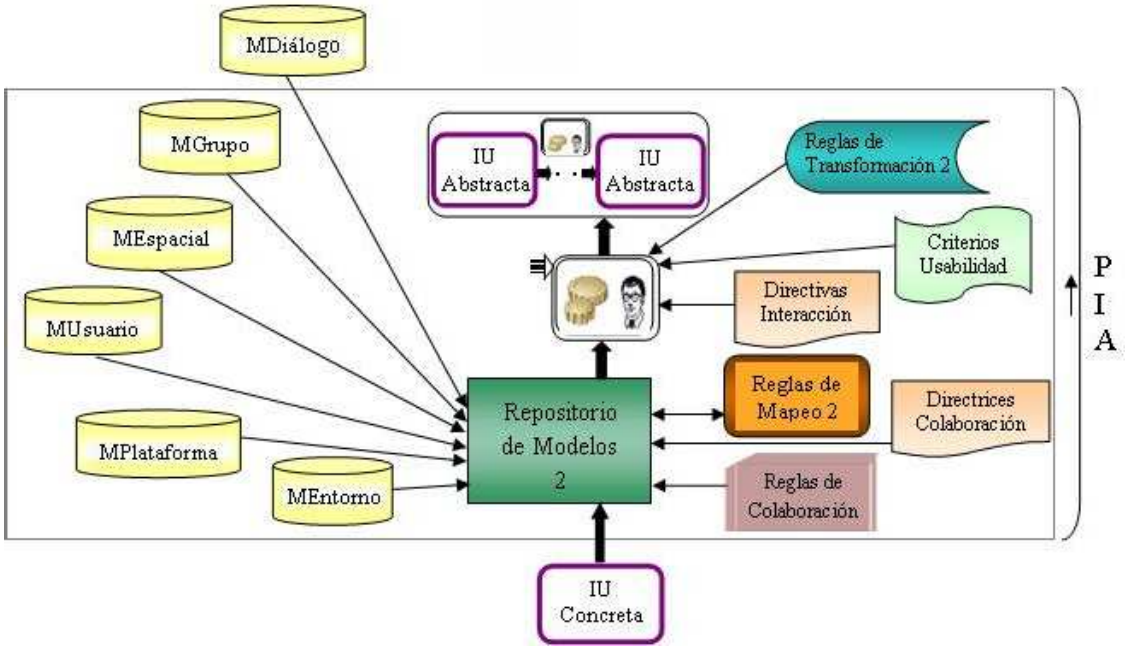


Figura 4.11: Fase *Proceso de Inferencia Abstracto*.

4.1.6. Activación del *Motor de Plasticidad Explícita*

Un *Motor de Plasticidad Explícita* –herramienta de generación de IUs plásticas- que opere bajo el enfoque de la *Visión Dicotómica de plasticidad* presentado en esta tesis, actúa inicialmente para producir la IU de partida y no vuelve a activarse hasta recibir una petición expresa por parte de la plataforma cliente con una descripción detallada de las restricciones del contexto de uso en curso (*restricciones de tiempo real*). Esto es lo que representa la entrada '*petición del cliente'* en la parte derecha de la figura 4.2. La petición recibida de la plataforma cliente dispara la operativa del servidor.

La activación del *Motor de Plasticidad Explícita* puede seguir diversos patrones dependiendo de la envergadura de la adaptación propia de cada situación, los cuales se relacionan en la *subsección 4.1.6.2*. No obstante, previo a la activación del proceso es indispensable proceder a la actualización de los *modelos contextuales*, con objeto de ajustar el *modelo de interfaz* a las condiciones del momento. Este proceso es el que se describe a continuación. La figura 4.12 (pág. 203) recoge todo lo relativo al proceso de actualización de los modelos y activación del *Motor de Plasticidad Explícita*.

4.1.6.1. Proceso de actualización de los modelos contextuales

Tras un periodo de utilización del sistema en la plataforma cliente, y tras haber afrontado diversas *variaciones dinámicas en el contexto de uso* (*Definición 2.5*), la nueva situación contextual en la que se encuentra en esos momentos el sistema interactivo (plataforma cliente) debe ser de alguna manera asimilada por el *servidor de plasticidad*. De otro modo no se conseguiría complementar la acción de ambos motores. Esta es la misión del proceso de actualización que se describe a continuación.

Tal y como ya se ha mencionado en la *sección 4.1.1.1*, el *enfoque de modelos compartidos* que se propone en este marco conceptual facilita la actualización de los *modelos contextuales* subyacentes, una vez transmitida la información contextual desde la plataforma cliente. En particular, el *repositorio de modelos 2* ejerce un papel esencial, como soporte para depositar y distribuir a los modelos las variaciones en los atributos del contexto en cada activación del *Motor de Plasticidad Explícita*. Como se observa en la figura 4.12, la información relativa a las *restricciones de tiempo real* (véase *Capítulo 2*; *sección 2.1.2.*), especificada en la petición enviada por la plataforma cliente, se deposita en el *repositorio de modelos 2* (conexión entre la *'petición del cliente'* y el *repositorio de modelos 2*), con el objetivo de ser redistribuida a los modelos implicados en la descripción del contexto de uso (*modelos contextuales*). La operación que representa esta actualización se denomina en este marco conceptual *introspección* (véase *sección 4.1.4.1.*). Sólo una vez actualizados los modelos podrá iniciar la generación de una nueva IU adecuadamente personalizada al contexto de uso, la cual puede seguir diversos patrones de activación, tal y como se presenta a continuación.

La actualización de los modelos constituye una facilidad adicional que no suele encontrarse en las herramientas basadas en modelos, y materializa un mecanismo de propagación de cambios contextuales, una de las limitaciones detectadas en la literatura, tal y como se expone en el *Capítulo 3* (véase *sección 3.2.5.*).

Existen dos modalidades para la operación de *introspección*, dependiendo de los modelos involucrados, así como de las circunstancias que hayan variado. A continuación se detallan las dos posibles alternativas de esta operación: por *notificación* y por *variación*.

4.1.6.1.1. Introspección por Notificación

Algunos modelos son notificados de una cierta circunstancia consistente en haber alcanzado una nueva situación cuando, bajo los términos manejados en cada caso, realmente se ha producido un cambio significativo (realización de una nueva tarea, ubicación del usuario

en una nueva localización geográfica, migración entre plataformas, cambio de usuario, cambio relevante de entorno). Por supuesto, si efectivamente se produce ese tipo de cambios, la aportación de los modelos correspondientes (*modelo de tareas*, *modelo espacial*, *modelo de plataforma*, *modelo de usuario* y *modelo de entorno*) en el proceso de construcción de la IU obviamente debe ser reemplazada por otra que se corresponda a la nueva situación. Para ser más precisos, la información a compartir a través de los *repositorios de modelos* debe ser substituida por la que corresponda a la nueva tarea, a la nueva ubicación del usuario, a la nueva plataforma, al nuevo usuario o a las nuevas características del entorno.

Así, los modelos susceptibles de ser notificados son éstos: el *modelo de tareas* y todos los modelos contextuales excepto el *modelo de grupo* (véase operación *Notifiación por Variación*): *modelo de usuario*, *de plataforma*, *de entorno* y *espacial*. Así, el *modelo de usuario* será notificado en el momento en que un nuevo usuario con un perfil distinto comience a hacer uso del sistema. El *modelo de plataforma* será notificado ante un cambio en la plataforma siendo utilizada (la *migración* entre plataformas podrá ser resuelta tanto de manera estática como dinámica). Por último, el *modelo de entorno* podrá ser notificado en los casos en que, al igual que en los dos casos anteriores, se trabaja bajo situaciones ambientales o temporales estereotipadas, esto es, ha sido estipulado que un conjunto de condiciones y factores relativos al entorno describen ambientes tipificados que se dan con cierta frecuencia. Por ejemplo, un ambiente cargado y ruidoso en un espacio cerrado y concurrido; o bien un espacio abierto, solitario y tranquilo pueden considerarse entornos estereotipo. En estos casos tiene sentido notificar al *modelo del entorno* de un cambio en la situación esteriotipada. En cuanto al *modelo espacial* será notificado cuando se produzca un cambio en la localización geográfica del usuario.

Este tipo de actualización queda plasmada en la figura 4.12 con la etiqueta 'N' de '*Notificación*'.

4.1.6.1.2. Introspección por Variación

En ocasiones los modelos van siendo en mayor o menor grado progresivamente configurados a partir de la información contextual proveniente de la plataforma cliente. Por supuesto, se trata de los *modelos contextuales*, con la excepción del *modelo espacial*, pues se supone que todas las ubicaciones a las que puede dirigirse el usuario están ya contempladas previamente en el modelo, y tan sólo se requiere ir notificando las localizaciones en las que se va encontrando.

La operación de actualización por *variación* es especialmente adecuada para el *modelo de grupo*, puesto que en principio es muy difícil tipificar posibles situaciones relativas al

grupo las cuales, además, serían completamente específicas para cada sistema colaborativo. Además, estas situaciones están sujetas a una continua variación. Por ello, en esta tesis se plantea este modelo como un modelo altamente versátil y variable que además, tal y como ya se ha mencionado en la *sección 4.1.2.1.2.*, pretende ser una representación del *conocimiento compartido* [CGPO02], tal y como se define en el *Capítulo 2* (véase *sección 2.1.3.*). En realidad, la información relativa al grupo debe irse literalmente construyendo a partir de la información que va siendo proporcionada por cada uno de los miembros del grupo. De ahí la importancia de un adecuado mecanismo de propagación de cambios, en especial en entornos colaborativos.

En el caso de los otros modelos (*usuario*, *plataforma* y *entorno*), la actualización se produce cuando se prevé la posibilidad de una *adaptación dinámica* en ciertos atributos que, más allá de ser considerados en la plataforma cliente, son, además, transmitidos al *servidor de plasticidad* para que también sean tomadas en consideración en la fase de reconfiguración de la IU las posibles *variaciones dinámicas en el contexto de uso*. Estas variaciones pueden involucrar tanto a lo que respecta al usuario (cambios en sus preferencias, necesidades o estado), al entorno (cambios en la condiciones, factores ambientales y temporales, como por ejemplo la intensidad de la luz ambiental) y a la plataforma (cambios en los aspectos tecnológicos de naturaleza dinámica -véase *Capítulo 2*; *sección 2.1.3.*-, como por ejemplo la capacidad de memoria disponible en un móvil en un momento dado).

Este tipo de consideraciones da soporte a los denominados *sistemas adaptativos* (definidos en el *Capítulo 2*; *véase sección 2.1.2.*), tanto desde la propia plataforma cliente (a través del *Motor de Plasticidad Implícita*), como también desde el *servidor de plasticidad*. De hecho, este aspecto ha sido planteado como una de las hipótesis de investigación en la presente tesis (véase *Hipótesis 5*; *Capítulo 1 - sección 1.3.*) para contribuir a la anticipación a los cambios contextuales, un aspecto especialmente problemático descrito en la literatura (véase *Capítulo 1*; *sección 1.2.* y *Capítulo 3*; *sección 3.2.5.*). El uso de un *enfoque de modelos compartidos* sirve de soporte para dar solución a esta limitación donde, por otro lado, es necesario que los modelos involucrados estén especialmente preparados para soportar variaciones.

Este tipo de actualización se plasma en la figura 4.12 con la etiqueta 'V' de '*Variación*'.

Como se ha podido observar en la explicación, y como también se desprende de la figura 4.12, parte de los *modelos contextuales* pueden ser objeto de actualización tanto por *notificación* como por *variación*. Se trata, en efecto, de los *modelos de usuario*, *de entorno* y *de plataforma*. Este aspecto queda reflejado en la figura 4.12 a través de la etiqueta N/V de '*Notificación/Variación*'.

4.1.6.2. Patrones de activación en la generación de una nueva IU

La gran flexibilidad aportada por la combinatoria entre las operaciones de *reificación* y *abstracción*, la posibilidad de iniciar el proceso en cualquiera de las seis fases, y por último, la posibilidad de que el diseño de la IU a un determinado nivel de abstracción pueda ser objeto de *adaptación* (*reglas de adaptación*) hace que los posibles patrones de activación se disparen en número.

Para simplificar su descripción, y considerando que son cuatro los posibles niveles de abstracción, se aplica la distinción en función del nivel donde se produce la activación del proceso de construcción de la IU, aspecto que está supeditado a la envergadura de la adaptación y de los aspectos involucrados. Se distinguen por tanto estas cuatro posibilidades:

1. Los modelos correspondientes al nivel superior de abstracción (habitualmente el *modelo de tareas*) deben variar para adaptarse a una nueva situación contextual, con lo cual se inicia el proceso en este nivel de abstracción. Este caso implica una activación de la fase 1 (*Preparación de los Modelos Iniciales*). Pueden darse dos modalidades para la obtención de un nuevo diseño de IU a este nivel de abstracción, según este paso consista en un paso de *abstracción* (estos modelos son inferidos a partir de una versión previa de la *IU Abstracta –reglas de transformación 1-*), o *adaptación* (los modelos en cuestión sufren un proceso de adaptación a un nuevo contexto de uso *–reglas de adaptación-*). Esta posibilidad queda representada en la figura 4.12 con el correspondiente *punto de entrada* (flecha flotante en la parte izquierda de las herramientas automáticas del nivel superior). En cualquiera de los casos la generación de la IU seguirá una derivación en *top-down*.

2. La *IU Abstracta* debe variar para adaptarse a una nueva situación. Pueden darse tres modalidades distintas para la obtención de un nuevo diseño de la *IU Abstracta*, según este paso consista en un paso de *abstracción* (la *IU Abstracta* es inferida a partir de una versión previa de la *IU Concreta -Proceso de Inferencia Abstracta-*), *reificación* (la *IU Abstracta* es derivada de los *modelos del nivel superior de abstracción –Proceso de Rendering Abstracto-*) o *adaptación* (la *IU Abstracta* sufre un proceso de adaptación a un nuevo contexto de uso). Las distintas posibilidades son reflejadas mediante los respectivos *puntos de entrada* en la figura 4.12. Como se admite un proceso multi-direccional, y dependiendo de la modalidad aplicada, la obtención de la *IU Abstracta* podrá desencadenar tanto una inferencia, con objeto de obtener nuevas descripciones de los modelos del nivel superior, como una derivación en *top-down* hacia la obtención de la *IU Final*.

3. La *IU Concreta* debe variar para adaptarse a una nueva situación. Al igual que en el caso anterior, pueden darse tres modalidades distintas para la obtención de un nuevo diseño de la *IU Concreta*, según este paso consista en un paso de *abstracción* (la *IU Concreta* es inferida a partir de una *IU Final -Proceso de Inferencia Concreta-*), *reificación* (la *IU Concreta* es derivada de la *IU Abstracta -Proceso de Rendering Concreto-*) o *adaptación* (la *IU Concreta* sufre un proceso de adaptación a un nuevo contexto de uso), aspecto reflejado con los respectivos *puntos de entrada*. Igualmente, como se admite un proceso multi-direccional, la obtención de la *IU Concreta* podría desencadenar tanto una inferencia, con objeto de seguir un proceso de *abstracción*, como un paso de derivación que daría lugar a una nueva versión de la *IU Final*.

4. El último caso por tratar es el que inicia el *Motor de Plasticidad Explícita* en la última fase (*Implementación*), con objeto de obtener una nueva *IU Final* partiendo de una versión previa de la *IU Concreta*. Se trata de un caso particular en que la *IU Final* puede ser generada directamente sin necesidad ni siquiera de adaptar la *IU Concreta*. Esta operación simplemente traduce una IU para una plataforma a la misma IU acomodada para ser operativa en otra plataforma distinta -aunque perteneciente a la misma familia de dispositivos-, traducción que comporta una mínima envergadura (no requiere el replanteamiento de nuevas tareas, ni tampoco una reconfiguración de los diseños de los niveles superiores de abstracción, ni tampoco variaciones importantes de renderización). En otras palabras, el *Motor de Plasticidad Explícita* también puede ser activado simplemente para llevar a cabo una recodificación de la *IU Concreta* a un lenguaje y a un conjunto de *widgets* adecuados para una nueva plataforma (intervención del *modelo de plataforma*), recurriendo al *visualizador* correspondiente. El *punto de entrada* en la fase de *Implementación* refleja esta posibilidad. Este caso ilustra un proceso de *migración dinámica*.

Cabe mencionar aquí la intervención de las *reglas de decisión 2*, unas reglas que son específicas de cada sistema, con objeto de decidir qué fase interesa disparar para una situación contextual dada. Por otro lado, las *reglas de decisión 1* ayudan también a la toma de decisiones previas al inicio de la fase de derivación/abstracción en que se haya determinado iniciar el proceso. Es por ello que se incluyen como componentes involucradas en la decisión acerca del patrón de activación y en el proceso de actualización de los modelos, al igual que ocurre con las *reglas de colaboración* y *directivas de colaboración*, que también influyen en estas decisiones.

La figura 4.12 (pág. siguiente) recoge los siete patrones de activación posibles, así como los componentes involucrados en (1) la toma de decisión acerca del patrón de activación a aplicar; y en (2) la actualización de los modelos.

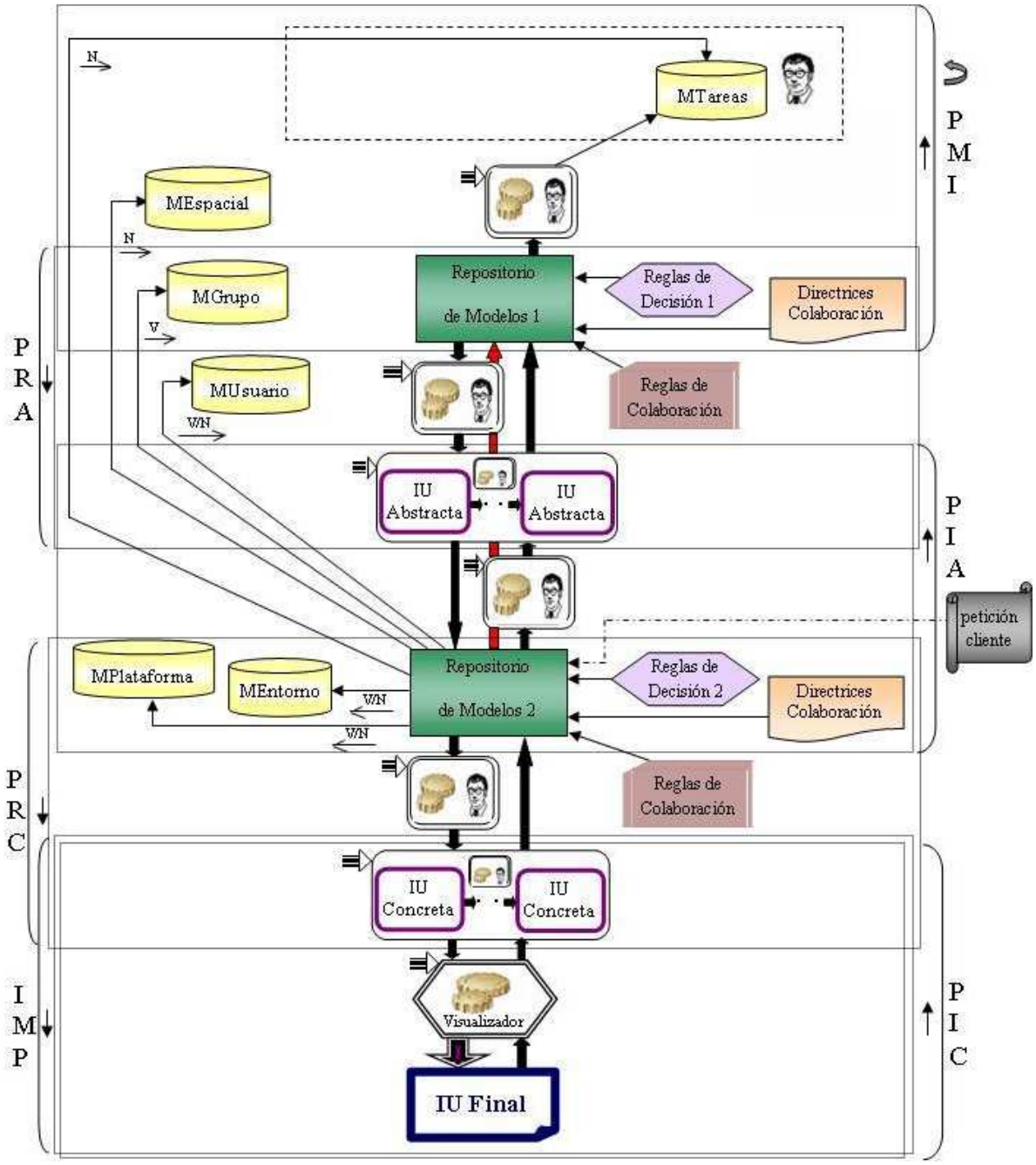


Figura 4.12: Componentes involucradas tanto en el proceso de actualización como en la decisión acerca del patrón de activación.

4.2. Estudio comparativo con el CAMELEON Reference Framework

El marco conceptual propuesto en esta tesis ha sido concebido para servir de orientación en la construcción de herramientas de generación de IUs que aplican el enfoque de la visión *dicotómica de plasticidad*. No obstante, permite representar tanto las herramientas que sigan este enfoque como las que no, puesto que constituye una extensión al *marco de referencia unificado* materializado en el CAMELEON Reference Framework [CCT+02a]. En esta sección se describen las diferencias principales entre ambos, así como los aspectos en los se aportan extensiones al ya existente.

4.2.1. Diferencias

Para empezar, se relacionan las diferencias entre el marco propuesto y el *marco de referencia unificado*. Sin ir más lejos, el *marco de referencia unificado* se describe como un marco para la especificación de IUs meramente multi-contextuales, y no para especificar IUs plásticas [CCT+03].

4.2.1.1. Enfoque subyacente

La primera diferencia, que puede considerarse la más destacable es el hecho de que el marco conceptual propuesto está acomodado a la aplicación del enfoque basado en la *visión dicotómica de plasticidad* (véase *Capítulo* 2; *sección 2.2*), tal y como ya se introduce en la *sección 4.1.1.1*. En efecto, este aspecto, cuya esencia consiste en separar y delimitar el tratamiento de las *adaptaciones proactivas* del tratamiento de las *adaptaciones estáticas* en dos motores distintos, dividiendo la infraestructura de plasticidad en dos mecanismos convenientemente realimentados (uno interno a la plataforma objetivo y otro externo, ubicado en el denominado *servidor de plasticidad*), repercute en gran medida en la definición de un marco de referencia que plasme el proceso de desarrollo y el método seguido bajo esta aproximación.

Así es, el enfoque basado en la *visión dicotómica*, que como ya se ha comentado, resulta novedoso en la literatura (véase *Capítulo 2*; *sección 2.2.3*), suscita un replanteamiento en la metodología, modelo de proceso y, en definitiva, en los marcos de referencia propuestos hasta la fecha, con objeto de ser adaptados a este nuevo modelo. En concreto, las repercusiones a que da lugar este aspecto son:

1. La ausencia de la parte encargada del procesamiento de las reacciones a desencadenar durante la ejecución. Esta parte queda delegada en la plataforma cliente a través del

Motor de Plasticidad Implícita (*Capítulo 6*). El CAMELEON Reference Framework incluye ambos motores, así como todas las consideraciones al respecto y modelos necesarios para llevar a cabo su especificación (*Modelo de Evolución* y *Modelo de Transición*, tal y como se propone en [DC03] -véase *Capítulo 3*; *sección 3.3.4.2*).

2. La necesidad de incorporar un mecanismo de propagación de los cambios experimentados en la plataforma cliente como consecuencia de la utilización del sistema y de su IU. En efecto, si de cada tipo de adaptación se encarga un motor distinto, es indispensable una adecuada realimentación entre los mismos, dado que bajo la *visión dicotómica* la IU sigue siendo personalizada al contexto de uso en tiempo de ejecución en la propia plataforma cliente. El objetivo perseguido es el de sincronizar la situación actual en la plataforma cliente con la información que del sistema y su uso se mantiene en el *servidor de plasticidad*, posibilitando no sólo la realimentación en ambos sentidos, sino incluso un mecanismo de anticipación a los cambios contextuales.

   Aquí es donde entran en juego tanto la petición explícita por parte de la plataforma cliente como todo el proceso de actualización de los *modelos contextuales* presentado. De hecho, la capacidad de transmitir los cambios producidos en la IU a los modelos subyacentes, que bajo el enfoque de la *visión dicotómica* cobra aún más importancia, constituye una limitación importante en la literatura. No hay que olvidar que, independientemente del enfoque aplicado, uno de los factores determinantes para la creación de una IU de calidad es su adecuación al contexto de uso al que va a ser destinada [Lóp05].

3. La tercera repercusión de la aplicación de la *visión dicotómica* es, precisamente, que se trata de un enfoque para proporcionar *plasticidad*. Este aspecto implica la necesidad de incorporar mecanismos que garanticen la preservación de la usabilidad a lo largo de todo el proceso de adaptación. Precisamente esa es la diferencia entre IUs plásticas e IUs multi-contextuales. Este aspecto se analiza a continuación (véase *sección 4.2.2.2*).

4.2.1.2. Incorporación de modelos, heurísticas y componentes

El marco conceptual propuesto incorpora nuevos modelos. Además, se propone plasmar el conjunto de componentes que intervienen en cada herramienta, para así poder realizar un estudio a un mayor nivel de detalle.

4.2.1.2.1. Modelos

En esta tesis se defiende que el contexto de uso, en todas sus facetas, debe ser especificado de manera comprensiva y detallada, haciendo uso de tantos *modelos contextuales* como se sea necesario. En este sentido, el marco conceptual propuesto por un lado (1) reutiliza los modelos existentes hasta ahora en el campo del diseño y desarrollo de IUs, con el fin de satisfacer los requisitos impuestos por la plasticidad; y por otro lado (2) introduce explícitamente nuevos modelos que hasta la fecha habían sido ignorados (*modelo de grupo*), o que no se habían utilizado de manera generalizada (es el caso, entre otros, del *modelo del entorno*, considerado imprescindible en entornos caracterizados por una movilidad, o bien por una constante variación en las circunstancias ambientales, y el *modelo espacial*, a considerar en *servicios basados en localización*), con objeto de representar el contexto de uso tal y como se propone y caracteriza en esta tesis (véase *Capítulo 2*; *sección 2.1.3*).

Otro modelo que no está explícitamente considerado en el CAMELEON Reference Framework es el *modelo de diálogo*. Inicialmente, la descripción de la interacción humano-ordenador, así como el estilo de navegación se describían de manera implícita a través del *modelo de tareas* y el de *dominio*. En general, el propio *modelo de tareas* es el que también especifica en muchas ocasiones cómo dirigir la operación de la IU. Por ejemplo, en la herramienta MASTERMIND [SSC+96], es el propio diseñador el que describe este aspecto mediante composiciones de las sub-tareas necesarias y sus interrelaciones, en un intento por suplir la falta de soporte de los aspectos dinámicos de las IUs, un problema conocido como "Main Window" [GFP+98]. Otro ejemplo es el caso de Teallach [GMP+98], donde la mayoría de los conceptos que habitualmente se capturan en un *modelo de diálogo* explícito se reparten entre los *modelos de presentación* y de *tareas*.

No obstante, la definición de estados de la IU y transiciones entre estados, así como la concreción de qué transiciones son posibles y qué no entre los mismos, juntamente con la secuenciación de la información, sin duda permite una descripción más detallada del estilo de navegación, así como una descripción exhaustiva del comportamiento dinámico de la IU. Actualmente es ampliamente reconocida la importancia de este modelo en la literatura [SE96b], y cada vez más es incorporado por las herramientas basadas en modelos, dado que su incorporación da lugar a la obtención de IUs más ricas que reflejan mejor las intenciones del diseñador [GFP+98]. Una de las primeras herramientas que incluyó un *modelo de diálogo* es TADEUS [ES95], el cual se genera automáticamente a partir de la especificación por parte del diseñador -mediante anotaciones en el *modelo de tareas*- de qué grupos de metas deben formar cada ventana. El correspondiente *modelo de diálogo* utiliza una notación llamada *grafos de diálogo* [SE96a], la cual especifica el comportamiento dinámico de la IU. Hoy en día existen multitud de técnicas que han sido propuestas para dar soporte a este modelo, tanto técnicas textuales como visuales. En el *Capítulo 3* se muestran las más utilizadas (véase *sección 3.2.6.3*).

4.2.1.2.2. Heurísticas y componentes

La especificación y representación de las reglas, directivas y bases de conocimiento que intervienen, así como la visualización de en qué fase participan, juntamente con una completa especificación de las dependencias y realimentación mutua entre componentes, sin duda fomenta un estudio en profundidad de las *herramientas basadas en modelos* representadas a través del marco conceptual propuesto. El manejo de todos estos detalles propicia una revisión exhaustiva con objeto de llegar a descubrir y plasmar el método seguido, el proceso de desarrollo y el enfoque aplicado en las herramientas objeto de estudio. Aunque en el CAMELEON Reference Framework se recomienda el uso de diversas componentes, éstas no aparecen integradas en el marco conceptual. En consecuencia, no existe la posibilidad de especificar qué herramientas las incorporan ni en qué fase actúan cada una de ellas y, como resultado, no se consigue 'retratar' la operativa subyacente. En definitiva, el nivel de profundidad alcanzado con la instanciación y posterior estudio de estas herramientas a través del *marco conceptual de referencia* no es el más óptimo.

Bajo la opinión del autor de esta tesis, poder representar más detalles acerca del funcionamiento, directrices y enfoques en los que se apoyan las herramientas estudiadas a través del marco conceptual proporciona un valor añadido. Es más, este punto de vista podría incluso ampliar la aplicabilidad del mismo, al ofrecer un instrumento de referencia suficientemente detallado como para describir en sus fases iniciales el prototipado a nivel conceptual de una determinada herramienta basada en modelos en vías de construcción.

La inclusión de heurísticas y recomendaciones también está en la línea de ofrecer orientación y guía en la especificación y explotación de modelos en el proceso de construcción de la IU.

4.2.2. Extensiones

Las bases del marco conceptual que se presenta en este capítulo aportan un valor añadido al *marco de referencia unificado* propuesto en el proyecto CAMELEON en varios aspectos que se detallan a continuación.

4.2.2.1. Consideraciones relativas al trabajo en grupo

Por primera vez se ha incorporado el tratamiento de la información relativa al desarrollo de una actividad grupal en un *enfoque basado en modelos*. Aunque a nivel de modelado existen diversas técnicas y notaciones, no existen propuestas acerca de cómo integrar esos

modelos y la información que se va generando relativa al grupo en un método *basado en modelos*.

Como es bien sabido, en el desarrollo de actividades colaborativas el objetivo perseguido con el uso del sistema no atañe a una sola persona, sino a un grupo, y por tanto la coordinación de esfuerzos entre los miembros integrantes es esencial para la buena marcha de la actividad. Sólo de ese modo se fomenta una colaboración real. Es por ello esencial disponer y manejar un entendimiento del *conocimiento compartido*, concepto definido en el *Capítulo 2* como *consciencia de conocimiento compartido* [CGPO02] (véase *sección 2.1.3*).

Tal y como se expone allí, los aspectos relativos al trabajo en grupo son considerados un aspecto más en la descripción del contexto de uso, constituyendo la cuarta componente en la caracterización del mismo. Como consecuencia, la visión del contexto de uso, hasta ahora entendida como una tupla formada por tres componentes (usuario, plataforma y entorno) [DC03] se amplía con un factor más a ser actualizado, considerado y explotado como parte integral del proceso de obtención de la IU. Aplicando el tratamiento oportuno a este nuevo *input* se pretende alcanzar ese entendimiento global del escenario grupal para que esa percepción también tome parte en el proceso de generación de IUs para entornos colaborativos. La meta perseguida en el proceso es la de producir IUs que, además de ser plásticas, estén personalizadas a las condiciones del trabajo en grupo, concepto definido anteriormente como *IUs sensibles al grupo* (véase *sección 4.1*). En efecto, construyendo IUs que satisfacen estas características se consigue que el entendimiento del *conocimiento compartido* llegue a repercutir y a plasmarse en la actividad de cada integrante del grupo, así como en el uso que éstos hagan del sistema dentro de un entorno colaborativo, evitando que se pierda en el *servidor de plasticidad*.

En particular, los aspectos clave para la incorporación y el tratamiento oportuno de esta nueva componente del contexto de uso han sido materializados en estas tres componentes: (1) la inclusión de una perspectiva global de las restricciones relativas al grupo de trabajo y el estado de la actividad grupal, que representa el *conocimiento compartido* [CGPO02], y que se especifica a través de un modelo no incluido hasta ahora: el *modelo de grupo*; (2) la especificación de las restricciones y requisitos relativos al sistema colaborativo en particular, a través de las denominadas *reglas de colaboración* (véase *sección 4.1.3.2* y *Recomendación 2 –sección 4.1.3.1*); (3) la inclusión de directrices reconocidas en el campo del *groupware*, y materializadas a través de la componente denominada aquí *directrices de colaboración* (véase *sección 4.1.3.1*).

Esta visión conjunta de la situación relativa al grupo no podría obtenerse si no se dispusiera de mecanismos para reunir las percepciones individuales de cada uno de los miembros, información que es mantenida y recopilada a nivel particular por cada plataforma

cliente participante, a través de los respectivos *Motores de Plasticidad Implícita* (*Capítulo 6*), y a la que se hace referencia como *consciencia de grupo particular* (véase *Capítulo 2*; *sección 2.1.3*). Así pues, la infraestructura de plasticidad propuesta en esta tesis bajo una arquitectura cliente-servidor, maneja la información relativa al grupo bajo dos perspectivas, una perspectiva global y una perspectiva local. El objetivo es proporcionar dos niveles de consciencia de grupo, el que se maneja a nivel individual por parte de cada miembro y el *conocimiento compartido*, que ofrece una visión centralizada y común como perspectiva global del grupo. Como consecuencia, deben ser proporcionados también los mecanismos y soportes necesarios para mantener también esa consciencia de grupo a nivel particular de cada miembro (véase definición de *mecanismos locales de consciencia de grupo* en *Definición 2.2*; *Capítulo 2*, *sección 2.1.3*), aspecto que será tratado en el *Capítulo 6*.

4.2.2.2. Evaluación de la usabilidad: diseño centrado en el uso

Otra de las recomendaciones incorporadas en el marco conceptual propuesto es la de combinar las prácticas propuestas para el *diseño centrado en el usuario* con el *diseño centrado en el uso* [CL99]. En este último caso el foco de atención para mejorar la usabilidad no está puesto en el usuario, sino en el uso (el trabajo que los usuarios se encuentran desempeñando y las tareas que están intentando llevar a cabo), utilizando métricas y métodos de diseño apropiados. En concreto, el criterio de calidad que más preocupa en el campo de generación de IUs plásticas es el de la usabilidad. La herramienta AB-UIDE [Lóp05] constituye un ejemplo de combinación de ambas técnicas. A la especificación de los criterios de usabilidad en AB-UIDE se le denomina *compromiso de usabilidad*, tal y como se expone en la *sección 4.1.3.1*. Algunas de las métricas de usabilidad aplicadas, enfocadas a obtener el máximo valor posible de usabilidad, son las métricas de diseño propuestas en [CL99]. Estas miden parte de los criterios de usabilidad planteados para un sistema determinado.

Otro aspecto a señalar al respecto de la usabilidad, el cual se tiene en cuenta en el *marco de referencia unificado*, y que también forman parte de la propia concepción de un entorno basado en modelos es la consideración explícita de diversas directivas de interacción que recogen la experiencia acumulada por los diseñadores de IUs, a fin de ser reutilizadas, tal y como se presenta en la *sección 4.1.3.1*. Una posible opción es la de incluir un módulo que actúe como asesor. Se trata de herramientas que analizan la información contenida en los diversos modelos declarativos, con objeto de verificar que el diseño satisface las propiedades especificadas de usabilidad, o bien produce estadísticas acerca de la calidad de la IU desarrollada y de la estructura del diálogo. Algunas de ellas son incluso capaces de simular la interactividad del usuario final.

4.2.2.3. Consideración de la tarea en curso en el proceso de derivación de la IU

El CAMELEON Reference Framework no considera la tarea en curso que el usuario está llevando a cabo como parte de la descripción del contexto de uso. Simplemente, esta información se deja implícita, aspecto que puede introducir limitaciones en las posibles adaptaciones soportadas por el marco conceptual.

Como se introduce en el *Capítulo 2* (véase *sección 2.1.3*), en el modelo propuesto en esta tesis la tarea constituye la quinta componente del contexto de uso a comunicar al servidor a través de una petición. También se ha mencionado en la *sección 4.1.6.1.* que el *modelo de tareas* es convenientemente notificado de cualquier cambio en la tarea en curso siguiendo el proceso de actualización de modelos previo a la reactivación del *Motor de Plasticidad Explícita*, operación que ha sido denominada *introspección por notificación*. De ese modo, la información relativa a la nueva tarea es oportunamente proporcionada al *repositorio de modelos 2* con objeto de ser convenientemente considerada durante todo el proceso de derivación de la IU, sin dejar de ser el centro de atención en todo momento.

Cabe señalar que la inclusión de este quinto atributo en la descripción del contexto de uso es esencial en un enfoque basado en la *visión dicotómica de plasticidad*, dado que entre dos situaciones posibles de *cambio contextual* notificadas al *servidor de plasticidad*, el proceso de interacción en la plataforma cliente puede haber avanzado substancialmente, y por consiguiente, existe la posibilidad de haberse producido no una, sino varias transiciones entre tareas. Es por ello que la pieza de información relativa a la tarea en curso constituye un elemento esencial a ser comunicado al *servidor de plasticidad*. No hay que olvidar que las tareas y el *modelo de tareas* que las representan constituyen el eje central en la mayoría de las herramientas basadas en modelos.

4.3. Instanciación del Marco Conceptual propuesto en distintas herramientas existentes

Tal y como se introduce al inicio de este capítulo, el objetivo del marco conceptual propuesto, denominado *Framework de Plasticidad Explícita Colaborativo*, es el de servir de instrumento de referencia para ayudar en la comprensión y razonamiento necesarios a la hora de estudiar las distintas herramientas de desarrollo de IUs basadas en modelos, denominadas en esta tesis *Motores de Plasticidad Explícita*. Con ese propósito, y tal y como se hizo en el *Capítulo 3* (véase *sección 3.3*), se han seleccionado un conjunto de herramientas basadas en modelos a fin de ser estudiadas, comparadas y representadas haciendo uso del marco conceptual propuesto.

4.3.1. Teallach

Como se ha ido mencionando a lo largo del capítulo, la herramienta Teallach [GBM+99] es la herramienta por excelencia que utiliza el *enfoque de modelos compartidos* adoptado en el marco conceptual propuesto, con el propósito de servir de almacenamiento común y de proporcionar un vehículo para facilitar la actualización de los modelos con los cambios producidos durante la utilización de la IU. De hecho, esta herramienta ha sido fuente de inspiración en todo momento para la formulación del marco conceptual presentado en esta tesis.

La característica más singular de Teallach es su flexibilidad, dado que permite al diseñador iniciar la construcción de una IU desde cualquier modelo, así como la transformación de conceptos entre cualquier combinación de modelos distintos de acuerdo a distintos caminos. Para hacer posible esta flexibilidad proporciona varios grupos de *reglas de mapeo* en distintos lugares de su arquitectura. Se trata, por tanto, de un método de desarrollo de IUs que explícitamente afronta el problema de desarrollo multi-direccional. En efecto, incorpora un enfoque transformacional, esto es, un mecanismo de transformación para mapear modelos entre sí. No obstante, a diferencia del enfoque transformacional basado en UsiXML, en este caso la lógica y definición de las *reglas de transformación* están completamente ligadas al código, con poco o ningún control por parte del diseñador. Por otro lado, la definición de estas representaciones no es independiente del motor de transformación.

Otra característica a destacar de Teallach es la posibilidad de intervención del experto humano, que es capaz de refinar o alterar la apariencia de la IU, definir la navegación entre ventanas y cuadros de diálogo o añadir extensiones no funcionales a la IU, a través de una herramienta de diseño. Además, una de las responsabilidades del *modelo de presentación* es la de actuar como repositorio de información de *Java Beans*[11], que son componentes software reutilizables que el *modelo de presentación* puede utilizar como bloques de construcción de la IU. Por lo tanto, podría asumirse esta biblioteca de Java Beans que se encarga de mantener el *modelo de presentación*, como la componente correspondiente a lo que en el marco conceptual propuesto se agrupa bajo la componente denominada *directivas de interacción*.

Los modelos utilizados en Teallach son: *modelo de tareas*, *de dominio* (representado como un diagrama de clases), *modelo de usuario* y *modelo de presentación*. En Teallach muchos de los conceptos que otras herramientas basadas en modelos capturan en un *modelo de diálogo* explícito están repartidos entre los *modelos de tareas* y *presentación*. Se

[11] Componentes de la arquitectura Java que se caracterizan por ser neutrales en cuanto a la plataforma. The Java Beans SSpecification. http://splash.javasolft.com/beans/docs/beans.101.ps

contemplan dos tipos de operaciones: (1) *enlazado* (*linkado*), o asociación entre componentes de dos modelos; (2) *derivación*, o construcción de los componentes de un modelo basado en los componentes de otro, aplicable a todos los modelos excepto en el de *dominio*. Una vez que los modelos han sido construidos y *enlazados* convenientemente tras un proceso iterativo de sucesivos refinamientos y transformaciones entre modelos, la IU final a ser entregada a la aplicación subyacente es generada a través del uso de un generador de código automático o *visualizador*.

A pesar de que no sigue un enfoque basado en la *visión dicotómica*, sí contempla, tal y como se ha comentado desde el inicio del capítulo, el flujo de información de la herramienta con el uso de la aplicación. Es por ello que recibe un feedback, el cual es adecuadamente tratado a través del uso de los repositorios de modelos. Esto se refleja en la figura 4.13, a través del *input* etiquetado 'utilización IU'.

En la figura 4.13 se representa el proceso descrito utilizando el marco conceptual propuesto (el *Framework de Plasticidad Explícita*).

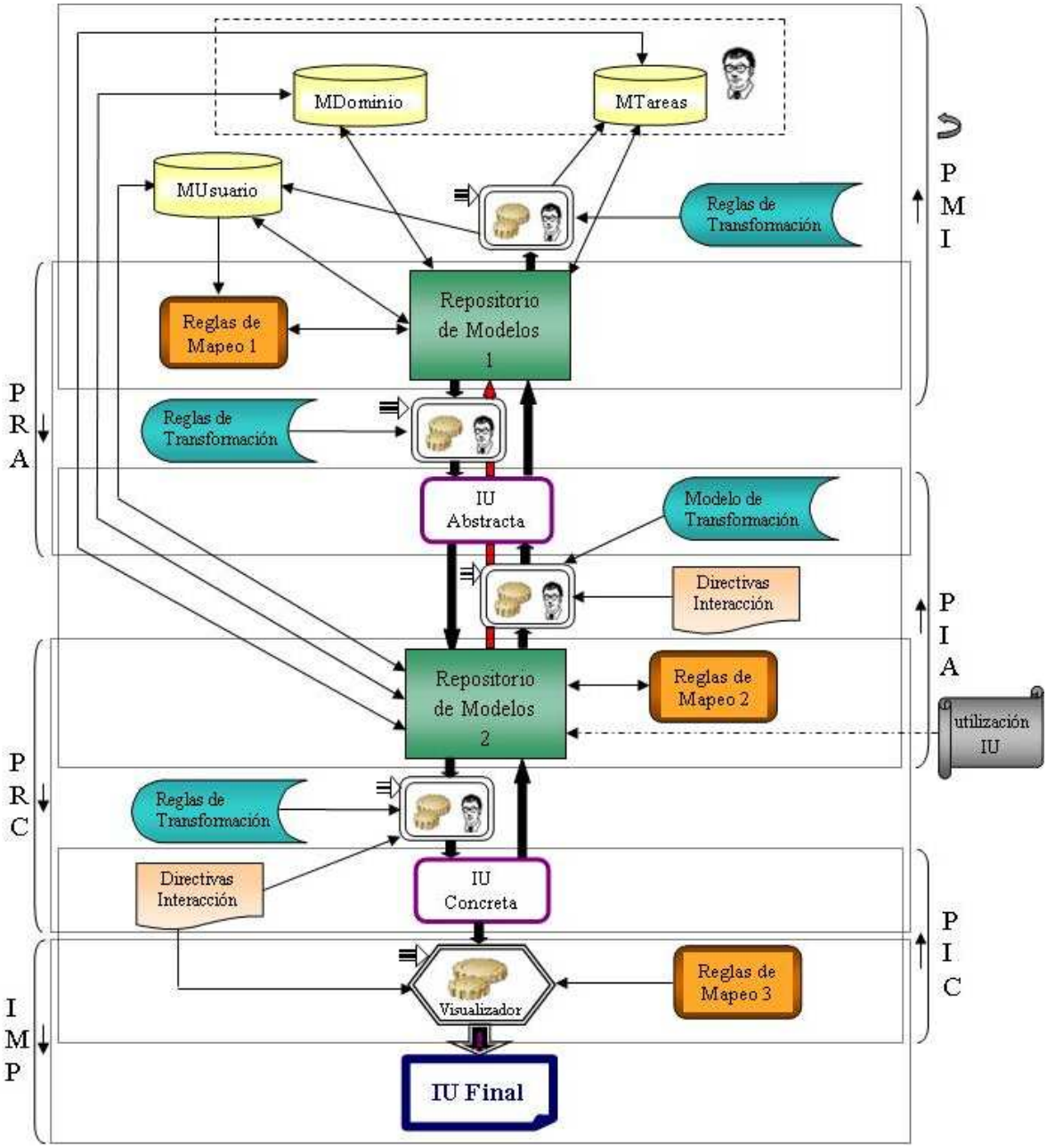


Figura 4.13: El marco conceptual propuesto instanciado en Teallach.

4.3.2. AB-UIDE

La principal característica de la herramienta AB-UIDE [Lóp05] es que a lo largo del proceso de diseño de la IU se persigue maximizar la calidad del producto final, en este caso la IU, integrando las técnicas oportunas. En este sentido puede considerarse como un método *centrado en el uso* [CL99]. Propone un método robusto basado en la transformación de modelos y guiado por las tareas que el usuario desea realizar con el sistema, las cuales son refinadas hasta conseguir una IU asociada a cada una de ellas. Tampoco renuncia a los beneficios de las aproximaciones *centradas en el usuario*. En esta línea incorpora el modelado del usuario, un entorno de interacción, así como un diseño iterativo basado en la evaluación de la IU mediante técnicas empíricas.

AB-UIDE permite también mantener la trazabilidad a lo largo del proceso de desarrollo, y aporta una notación gráfica para su visualización y edición de una forma clara e intuitiva para el diseñador a través del concepto de *conector*[12]. El modelo de conectores propuesto en [LMM+03], [LMFL03], los cuales son utilizados en la descripción de arquitecturas software, es el método utilizado para representar las relaciones establecidas entre los distintos modelos (*modelo de mapping* en AB-UIDE –figura 4.14-), y disponer de ellas en tiempo de ejecución, con objeto de poder llevar a cabo un proceso de adaptación más exacto.

La herramienta AB-UIDE no aplica la *visión dicotómica* propuesta en esta tesis, es decir, no existe una delimitación en dos motores distintos para llevar a cabo las *adaptaciones proactivas* y las *adaptaciones estáticas* por separado. Todas ellas se llevan a cabo a través de la herramienta utilizando un método que se apoya en un sistema multi-agente. Así, una serie de agentes colaboran inteligentemente para proporcionar al usuario la adaptación más adecuada para cada situación presentada durante la interacción con la IU. En este caso, por tanto, se produce una realimentación de los modelos subyacentes mediante la comunicación con los agentes de la IU.

Eso es lo que se refleja en la figura 4.14 a través de la componente etiquetada con '*comunicación agente IU*', que aparece conectada con los modelos contextuales y con el *modelo de tareas*. Esta realimentación hace posible la actualización de los modelos. En particular, y tal y como se plasma en la figura 4.14, se aplica *Introspección por Notificación* en el *modelo de tareas* (en particular este modelo puede ser objeto de adaptación –flecha entrante en la fase *Preparación de los Modelos Iniciales*-), por *Variación* en el *modelo del entorno*, y ambas modalidades en el *modelo de usuario* y el de *plataforma*.

[12]Se trata de un conjunto de roles y una especificación de un pegamento que los mantiene unidos [AG97]. Los roles modelan el comportamiento de cada parte involucrada en la interacción, mientras que el pegamento proporciona la coordinación entre las instancias de cada rol [WLF00].

La figura 4.14 representa la herramienta AB-UIDE utilizando el marco conceptual propuesto.

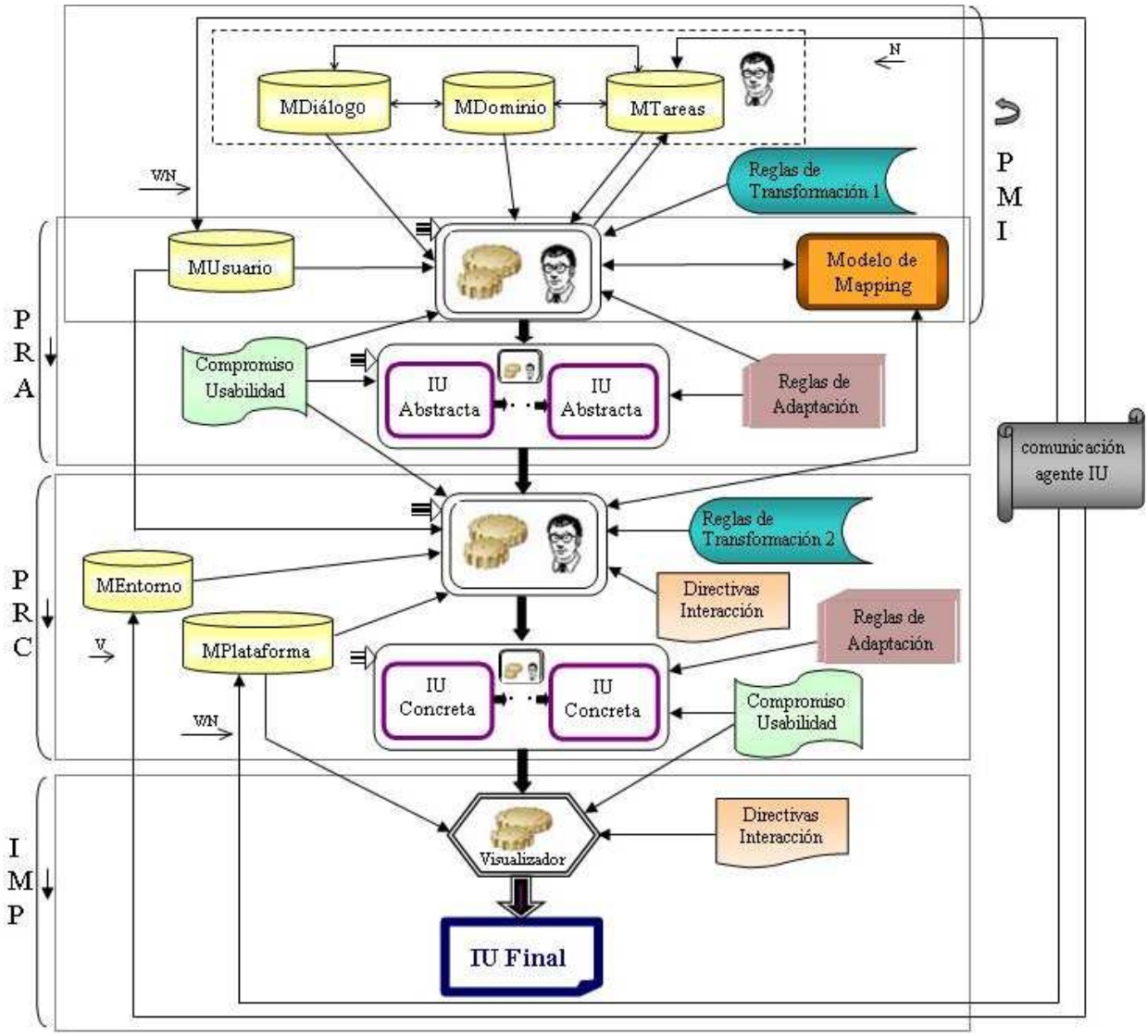


Figura 4.14: El marco conceptual propuesto instanciado en AB-UIDE.

AB-UIDE sigue un enfoque transformacional. El proceso seguido para llevar a cabo tanto la transformación entre modelos como la adaptación de un modelo determinado a otra versión del mismo sigue un enfoque de reescritura de gramáticas de grafos, descrito en [LVM+04]. Todo el proceso de transformación se sustenta sobre el lenguaje de definición de IUs usiXML [LVM+04], el cual ha sido enriquecido para soportar la plasticidad de manera más apropiada.

Algunas de las técnicas y lenguajes más relevantes utilizados para algunas de sus componentes han sido ya mencionadas a lo largo de este capítulo. A continuación se recopilan las más destacadas.

- *Modelo de Tareas.* Para la especificación de este modelo se conjugan tres notaciones: los diagramas de estado [Har87], los ConcurTaskTrees [Pat99] y las herramientas abstractas de interacción [Con03].

  Las tareas se describen mediante un nombre único, una descripción para la documentación del modelo, la frecuencia con que se espera se ejecutará la tarea/acción, y una precondición y postcondición expresadas en OCL[13] (*Object Constraint Language*).

- *Modelo de dominio.* Utiliza el diagrama de clases de UML. Con objeto de maximizar la utilidad de este modelo, proporciona una descripción detallada de los tipos de datos involucrados tanto en los atributos de las clases como en los métodos.

- *Especificación del diálogo.* Se utilizan las herramientas abstractas propuestas por Constantine en [Con03], donde se presenta un conjunto de operaciones que pretenden ser un corpus de constructores para la especificación abstracta de las acciones básicas que lleva a cabo un usuario a través de la IU. En AB-UIDE la especificación del diálogo se realiza al mismo tiempo que la especificación de las tareas, ya que consiste en ir decorando esta última mediante el etiquetado de las transiciones entre las tareas con las herramientas abstractas de [Con03].

- E*specificación de la plataforma.* Utiliza el estándar CC/PP[14] (véase *Capítulo 2*; *sección 2.1.6.2.*).

- *Modelos transitorios.*

  - *IU Abstracta.* Se utilizan los *objetos abstractos de interacción* propuestos por UMLi en [Pin02], enriquecidos con un nuevo tipo de *objeto abstracto de interacción*: el denominado *selector*. La *IU Abstracta* se almacena en usiXML.
  - *IU Concreta.* Utiliza el modelo concreto de IU propuesto para usiXML. Este modelo recoge de manera abstracta, pero a su vez suficientemente concreta, todos los *widgets* disponibles en la mayoría de las plataformas destino. Además, también permite especificar el posicionamiento de los elementos dentro de los

[13] http://www.klasse.nl./ocl
[14] http://www.w3.org/Mobile/CCPP

contenedores utilizando un modelo de cajas como el usado en el lenguaje XUL[15] o la distribución de componentes *BoxLayout* del lenguaje Java[16].

- *Producción de la IU Final.* Se genera a través de generadores automáticos de código (*visualizadores*). Los *visualizadores* disponibles hasta el momento soportan la generación de la IU para Java (la modalidad J2SE), Java para dispositivos móviles (J2ME) y XUL+Javascript.

- *Especificación de los criterios de usabilidad.* Como ya se ha mencionado, en AB-UIDE esta componente recibe el nombre de *compromiso de usabilidad*. Describe el peso y la influencia que cada criterio de usabilidad debe tener en la generación de la IU. Se ha utilizado una variante de *Goal-oriented Requirement Language*[17] (GRL) que permite al diseñador capturar los requisitos del *compromiso de usabilidad*, así como su documentación. Además, se aplican las métricas de diseño propuestas en [CL99] para medir parte de los criterios de usabilidad planteados para un sistema determinado. Para ser más precisos, con objeto de preservar al máximo la usabilidad de la IU generada, el sistema compara las propiedades de usabilidad en la IU original con aquellas de la IU adaptada, con objeto de comprobar si se preserva la usabilidad.

- *Directivas de interacción.* Como *directivas de interacción* utiliza patrones de interacción como los propuestos en [Wel04], [Tid02] y [MLML03], asociándolos a los casos de uso identificados durante la fase de adquisición de requisitos.

Aunque dentro del método se proponen reglas para la traducción de las combinaciones de objetos abstractos más comúnmente utilizados, se deja el camino abierto al diseñador para que añada sus propias reglas. Como la *IU Abstracta* se almacena también en usiXML, el diseñador puede crear nuevas reglas de transformación usando cualquiera de los procedimientos habituales aplicables a la transformación de una especificación XML: transformaciones XSLT[18], transformaciones de gramáticas de grafos [Lim04], o transformación de especificaciones algebraicas [BCR05].

En cuanto a la flexibilidad ofrecida se puede mencionar que admite la combinación de dos operaciones: la *reificación* vertical y la *adaptación*. Además, permite iniciar el proceso de derivación en cualquiera de las fases de abstracción. Es por ello que aparecen puntos de entrada en todos los niveles excepto en el último, puesto que no soporta la operación de *abstracción* (ingeniería inversa). Por último, permite la intervención del humano en todas las fases.

[15] http://www.xulplanet.com
[16] http://java.sun.com
[17] http://www.cs.toronto.edu/km/GRL/
[18] http://www.w3.org/TR/xslt

4.3.3. TERESA

TERESA (Transformation Environment for inteRactivE Systems representAtions) [PS03], [MPS03], presentada en el *Capítulo 3* (véase *sección 3.4.1.*), es la herramienta para el diseño de IUs multi-dispositivo desarrollada en el proyecto CAMELEON. Se trata de una herramienta orientada a la Web diseñada para la generación de IUs para distintas plataformas móviles en fase de diseño. Se basa en el refinamiento del *modelo de tareas* general obtenido para un ordenador de sobremesa, mediante la aplicación de diversas técnicas de presentación y de diálogo, con objeto de transformar los refinamientos obtenidos en código XHTML en descripciones lógicas de la IU válidas para distintas plataformas móviles (PocketPC y teléfonos móviles).

Al igual que las herramientas presentadas anteriormente, sigue un enfoque transformacional. Aunque la versión inicial de TERESA sólo permite la operación de reificación vertical, así como la traducción aplicada únicamente sobre el *modelo de tareas*, en la versión posterior presentada en [CMP04] se introducen nuevas capacidades de transformación en el proceso, con objeto de soportar un ciclo de desarrollo mucho más flexible, tal y como se explica en el *Capítulo 3*. Son las que se resumen a continuación: (1) se introduce la posibilidad de adaptar la *IU Abstracta*; (2) se añade un punto de entrada en los niveles de la *IU Abstracta* e *IU Concreta* (eso permite ignorar el nivel superior de abstracción); (3) la habilidad para transformar el diseño para distintas plataformas móviles en niveles intermedios de abstracción, a través de un módulo de rediseño, el cual aparece enmarcado en la figura 4.15 mediante la etiqueta 'MOD REDI'; y (4) el rediseño de una IU ya existente para adecuarlo a otra plataforma, partiendo del nivel concreto.

Aunque en esta última versión es posible introducir las IUs resultantes de aplicar ingeniería inversa a través de la herramienta *Vaquita* [BVS02], para la generación de nuevos diseños no está contemplada como parte integrante de la herramienta. No obstante, en el soporte para el rediseño proporcionado en esta versión [CMP04], sí se contempla la transformación de la *IU Concreta*, juntamente con alguna información adicional del nivel abstracto, en una descripción tanto para el nivel abstracto como para el nivel concreto, válidas para el dispositivo móvil destino, a partir de las cuales generar automáticamente la *IU Final*. Es por ello que en la figura 4.15, que muestra una representación gráfica de esta versión más reciente de TERESA a través del marco conceptual propuesto en este capítulo, aparece una flecha ascendente entre la *IU Concreta* y la *IU Abstracta*.

Al ser una herramienta estrictamente concebida para la generación de IUs para distintas plataformas móviles en fase de diseño, es decir, que aborda la *migración estática*, no requiere un soporte para la fase de ejecución, como ya se refleja en la descripción de la misma a través del *marco de referencia unificado*, realizada en el *Capítulo 3* (véase *sección*

3.4.1.). En consecuencia, los únicos cambios contextuales que soporta son los cambios en la plataforma destino. Es por ello que no recibe ningún feedback correspondiente a la fase de ejecución, tal y como se observa en la figura 4.15. La única información que requiere para reiniciar el proceso es la referente a la plataforma destino y la IU ya existente con la que iniciar el proceso de transformación. En cualquier caso, esa información corresponde a la fase de diseño. En consecuencia, no sigue el enfoque de la *visión dicotómica* propuesta en esta tesis, ni tampoco es necesario un proceso de actualización de los modelos. Todas las operaciones aplicadas consisten en operaciones de *reificación* y adaptación (*traducción*). Tampoco se soporta en el *enfoque de modelos compartidos* para llevar a cabo el proceso de transformación. En la figura 4.15 se representa la herramienta TERESA utilizando el *Framework de Plasticidad Explícita*.

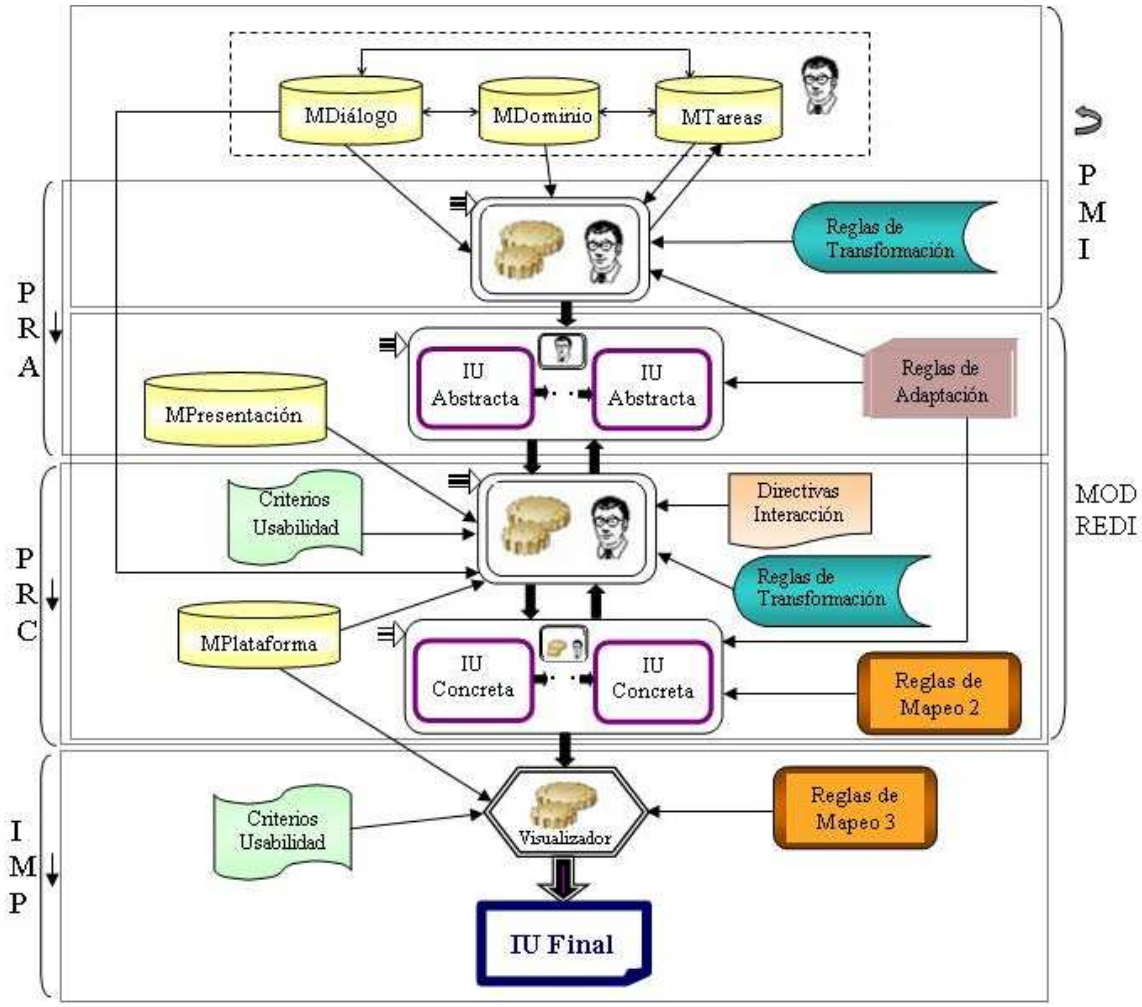


Figura 4.15: El marco conceptual propuesto instanciado en TERESA.

4.3.4. DynaMo-AID

DynaMo-AID (**Dyna**mic **Mo**del-b**A**sed user **I**nterface **D**evelopment) [CLC04a] proporciona un soporte de diseño y una arquitectura de ejecución para IUs sensibles al contexto que soporta la adaptación no sólo a la plataforma, como ocurre en Dygimes, sino también a las condiciones del entorno, tal y como se introduce en el *Capítulo 3* (véase *sección 3.4.3.*). De hecho, DynaMo-AID constituye una parte de la herramienta Dygimes [Luy04] que complementa la parte de sensibilidad al contexto.

Una de las particularidades más significativas de DynaMo-AID es la combinación de la información contextual en los distintos niveles de desarrollo de la IU, extendiendo los modelos tradicionales para poder incorporar una componente del contexto y de este modo soportar el diseño de este tipo de IUs, así como la consideración del diseño de una IU para un servicio cuando éste pasa a estado disponible para la aplicación. Este último punto es la principal contribución de la herramienta. En [CLC04b] se presenta la manera como estos modelos son extendidos. La idea fundamental consiste en definir versiones dinámicas de cada tipo de modelo (*modelos dinámicos*), los cuales pueden cambiar en tiempo de ejecución, hasta incluso fusionarse con otros modelos del mismo tipo. Así, por ejemplo, el *modelo de tareas dinámico* en DynaMo-AID es un ConcurTaskTrees al que se le ha añadido información relativa a la plataforma y al contexto.

El generador de código que ofrece es capaz de transformar dinámicamente una especificación de tareas en una IU operativa, haciendo uso de algoritmos que se incluyen en la propia herramienta que siguen un proceso de *reificación* vertical. Por lo tanto tan sólo incluye un punto de entrada en el nivel de tareas, y la intervención del humano es tan sólo puntual. Otros detalles del proceso de desarrollo de IUs a través de DynaMo-AID se detallan en el *Capítulo 3*.

Como se trata de una herramienta de soporte a la ejecución de IUs sensibles al contexto, constantemente va recibiendo información, no sólo relativa a los cambios contextuales, sino también la relativa a los nuevos servicios que van apareciendo. Al no seguir el enfoque de la *visión dicotómica*, el mismo motor resuelve cualquier tipo de adaptación, en tiempo de ejecución. Por lo tanto, no se requiere una petición por parte del cliente. El *input* correspondiente se representa con la etiqueta *'nuevos servicios + información contextual'*. Esos cambios deben repercutir en los modelos, los cuales deberán ser actualizados (operación de *introspección*), o incluso adaptados (operación de *adaptación*). Es el caso del *modelo de entorno dinámico* y del *modelo de diálogo dinámico*, donde es en este último caso el experto humano interviene en la operación de *adaptación*. Por su parte, como existen referencias entre estos modelos y el *modelo de tareas*, este también sufre la correspondiente

adaptación al contexto de manera indirecta. Como no sigue el *enfoque de modelos compartidos*, no se vale del uso de *repositorios de modelos* para centralizar la distribución de todas estas actualizaciones y adaptaciones a los respectivos modelos.

Todo ello queda reflejado en la figura 4.16, donde se refleja el método seguido en el proceso de derivación de DynaMo-AID a través del marco conceptual propuesto.

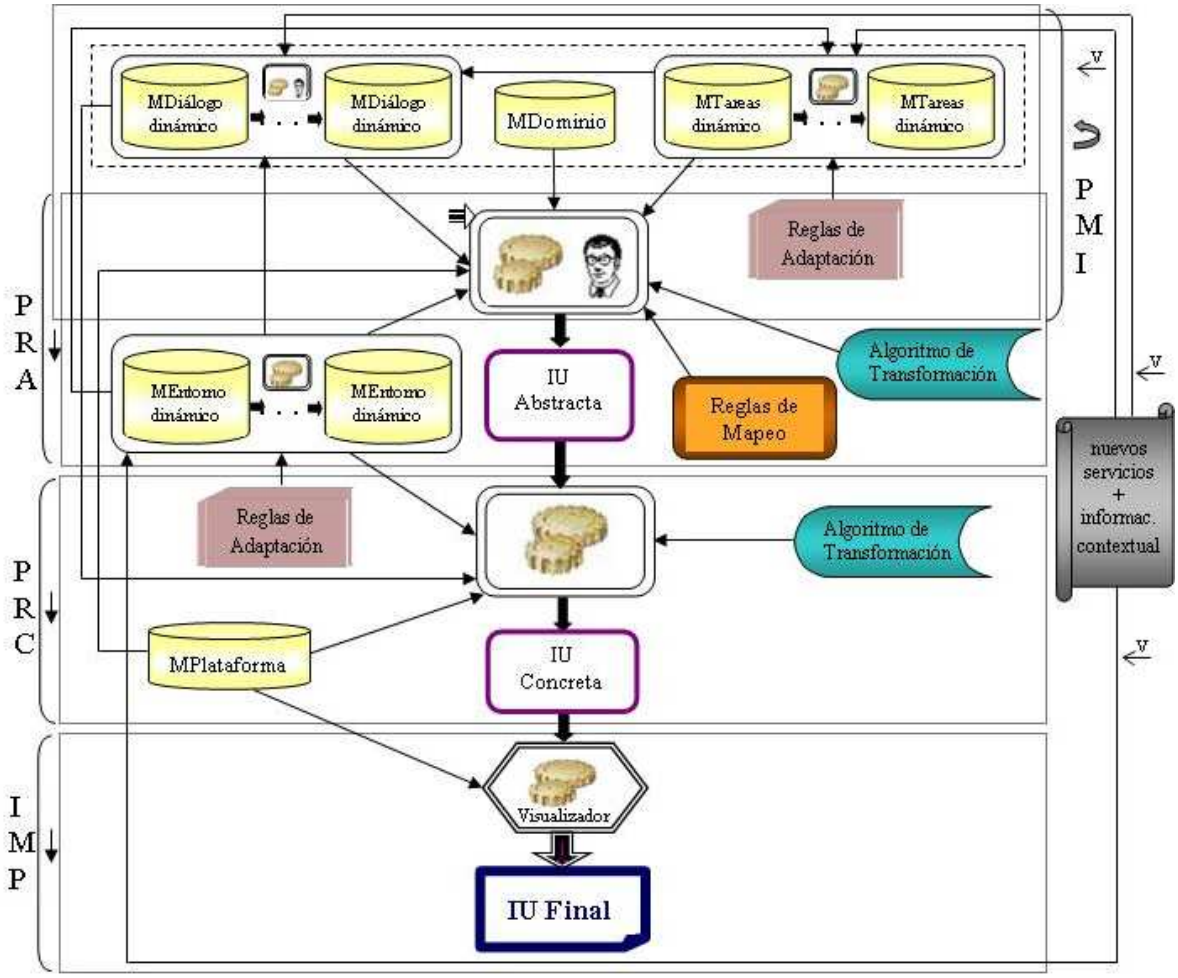


Figura 4.16: El marco conceptual propuesto instanciado en DynaMo-AID.

4.4. Resumen y conclusiones del capítulo

El enfoque MDA (Model-Driven Architecture) [Bro04] en general, y el *enfoque basado en modelos* en el mundo de la interacción constituyen en la actualidad la piedra angular en la investigación para la especificación y desarrollo de sistemas software. Están concebidos

con el objetivo principal de separar el diseño de la aplicación, que alberga los requisitos funcionales de la misma, de la arquitectura y tecnologías utilizadas para su construcción, es decir, de la infraestructura que hace efectivos los requisitos no funcionales. En efecto, el uso de modelos favorece que el desarrollador centre su foco de atención en el *'qué'* de lo que debe ser representado, más que en el *'cómo'* debería ser representado. Centrándonos en el caso de las herramientas basadas en modelos, el diseño de la IU se concibe como un proceso de creación y refinamiento del *modelo de interfaz*. Mientras la sofistificación de la IU generada depende en gran medida de las capacidades y expresividad del *modelo de diálogo* y el de *presentación*, la funcionalidad subyacente de la IU es principalmente un reflejo de la riqueza de los *modelos de dominio* y de *tareas*.

Sin renunciar a los beneficios que las aproximaciones centradas en el usuario proporcionan al desarrollo de la IU, la combinación de las prácticas propuestas en este campo con la aplicación de métodos de evaluación de usabilidad hace que el producto final pueda calificarse también como un *diseño centrado en el uso* [CL99], puesto que se trabaja en la línea de asegurar la calidad. Esta es una de las aportaciones del marco conceptual que se propone en este capítulo, como marco teórico general, concebido para razonar acerca del proceso de generación de IU plásticas. Sin embargo, el aspecto más destacado es que se contempla por primera vez la base teórica para el tratamiento de las necesidades intrínsecas de los entornos colaborativos.

Hoy en día observamos que, efectivamente, cada vez es más habitual el uso de dispositivos móviles en entornos de trabajo en grupo, como consecuencia de la penetración de la computación móvil también en el campo de *groupware*. En este sentido se ha procurado proporcionar un marco conceptual que integre y soporte los aspectos de movilidad, adaptación y heterogeneidad de dispositivos en este tipo de entornos colaborativos noveles, caracterizados por una distribución tanto espacial como temporal, una heterogeneidad y un dinamismo. La clave para la incorporación del grupo es la de integrar y engranar los componentes de plasticidad con ciertos componentes de colaboración, haciendo que las directrices más importantes del campo de groupware tomen un papel principal en el proceso de desarrollo de IUs. En definitiva, se trata de reutilizar la infraestructura de plasticidad para explotar también la información de consciencia de grupo, enriqueciendo de ese modo el proceso de adaptación subyacente. La meta perseguida es la de ofrecer un soporte que fomente la colaboración, como pieza clave del desarrollo *groupware*, proporcionando al mismo tiempo la flexibilidad perseguida por la plasticidad.

Otros aspectos destacables del marco conceptual propuesto son: (1) la gran flexibilidad proporcionada para la descripción de cualquier tipo de herramienta basada en modelos; (2) su exhaustividad al considerar, a diferencia del *marco de referencia unificado*, no sólo los modelos que intervienen, sus dependencias y operaciones soportadas, sino también

cualquier tipo de componente adicional. Como consecuencia, el 'retrato' obtenido de las herramientas estudiadas a través del marco conceptual ofrece un conocimiento más a fondo del enfoque y de las técnicas empleadas; y, por supuesto, (3) la adecuación del marco conceptual al enfoque de la *visión dicotómica* propuesto en esta tesis. Este último aspecto ha requerido afrontar dos problemas reconocidos en el ámbito de las herramientas basadas en modelos: la propagación y la anticipación a los cambios contextuales.

La instanciación del marco conceptual propuesto en una herramienta operativa requiere (1) concretar y formalizar los modelos y técnicas empleadas en la herramienta objeto de estudio; (2) estructurar el proceso de construcción de IUs seguido. En este sentido se proporcionan no sólo diversas opciones o sugerencias encontradas en la literatura para la implementación de algunos de los componentes, sino también un conjunto de heurísticas preeliminares y recomendaciones con el propósito de guiar al desarrollador en su concreción y, en consecuencia, facilitar el despliegue de *Motores de Plasticidad Explícita* concretos. Algunas de estas heurísticas constituyen una recopilación de las observaciones y recomendaciones encontradas en la literatura. Otras son fruto del análisis y deducciones a las que se ha llegado como consecuencia de la elaboración de la presente tesis.

Por último, no hay que olvidar que bajo el enfoque de la *visión dicotómica*, la salida obtenida por el *Motor de Plasticidad Explícita* (la IU generada para satisfacer las *restricciones de tiempo real*) sirve como entrada al *Motor de Plasticidad Implícita* ubicado en la plataforma cliente, y cuyas capacidades de adaptación van más allá de los modelos diseñados durante el proceso de diseño, con objeto de ofrecer adaptaciones proactivas.

Parte III

Plasticidad Implícita

Capítulo 5

Estado del arte en sistemas con *Plasticidad Implícita*

"The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are undistinguishable from it."

Mark Weiser."The Computer for the 21st Century"

Desde principios de los años sesenta, el concepto de contexto ha sido modelado y explotado de diversas formas y en muy diversas áreas. En concreto, en el área de la *Computación Sensible al Contexto* la comunidad científica ha redescubierto este concepto, debatiéndolo durante años sin alcanzar un consenso acerca de su definición operativa. Debido a la falta de teorías predictivas para establecer el uso apropiado del contexto, se ha adoptado un enfoque empírico que ha dado lugar al desarrollo de gran cantidad de aplicaciones a pequeña escala utilizadas como prototipos y pruebas del concepto. Existen numerosos grupos de investigación trabajando en el campo de la *computación sensible al contexto*, por lo que resulta necesario adquirir un cierto conocimiento acerca de lo que se está tratando de alcanzar con sus desarrollos.

En este sentido, en la primera sección se caracteriza el área de estudio, equiparando el concepto de *plasticidad implícita* con el de *sensibilidad al contexto*. A continuación se presenta el trabajo relacionado organizado por categorías. Esta revisión comienza presentando diversos grupos de *aplicaciones sensibles al contexto* con el propósito de analizar sus limitaciones y problemática y tratar de determinar qué requisitos debe cumplir una infraestructura más general que facilite su desarrollo. A continuación se centra la revisión del estado del arte en el estudio de aquellas herramientas que tienen como objetivo soportar el desarrollo de *aplicaciones sensibles al contexto*. Para finalizar se muestra una comparativa de las infraestructuras analizadas en el capítulo.

5.1. Caracterización del área de estudio

Definimos un sistema con *plasticidad implícita* como un sistema que es capaz de percibir y/o inferir información contextual, utilizándola a su favor para conseguir adaptar la IU o bien ajustar el comportamiento y/o aspectos concretos de la funcionalidad de la aplicación a distintos contextos de uso de manera automática.

Bajo la opinión personal del autor de esta tesis, el concepto de *plasticidad implícita* se equipara con el de *computación sensible al contexto* (concepto definido en detalle en el *Capítulo 2 -sección 2.1.3-*) siempre y cuando (a) en las propiedades que caracterizan el contexto se incorpore *cualquier tipo de información disponible que pueda ser relevante para la aplicación* pudiendo abarcar, entre otros, el propósito de la *personalización* (véase *Capítulo 2*; *sección 2.1.6*) y distintos aspectos del trabajo en grupo; y (b) la aplicación de las adaptaciones ante los cambios contextuales se realiza preservando en todo momento la usabilidad. En particular, la caracterización del contexto utilizada en esta tesis se presenta en el *Capítulo 2* (véase *Capítulo 2*; *sección 2.1.3*).

La *computación sensible al contexto* es un área de investigación relativamente nueva que utiliza la información del contexto para mejorar la operativa de las aplicaciones. Las aplicaciones que satisfacen esta funcionalidad son denominadas *aplicaciones sensibles al contexto*. Una buena definición de *aplicación sensible al contexto* es la siguiente:

> *"aplicación que utiliza información acerca del contexto para mejorar su operativa o proporcionar información relevante y/o servicios al usuario, donde la relevancia está supeditada a la tarea del usuario"[Ens02].*

Este tipo de aplicaciones puede abordar multitud de situaciones contextuales en las que confluyen múltiples parámetros cambiantes. En concreto, las consideraciones acerca de los recursos hardware adquieren hoy en día vital importancia, tanto por sus amplias posibilidades de elección y potencialidad con las que vienen provistos como por la necesidad implícita de interconexión entre dispositivos. Este último caso hace referencia al campo de la *Computación Ubicua*. En [Sen07] se presenta una caracterización de la *computación ubicua* y una revisión del estado del arte, así como también una categorización de los distintos términos y sus relaciones con otros campos muy próximos.

A pesar de ser un área de investigación activa, tal y como se comenta en el *Capítulo 2* (véase *Capítulo 2*; *sección 2.1.2.*), no existe un entendimiento común acerca de lo que es el contexto, ni de cómo puede ser utilizado para mejorar la operación de las aplicaciones. Uno de los problemas con los que debe enfrentarse un desarrollador de *aplicaciones sensibles al contexto* es el problema de obtener la información contextual. Una arquitectura del

contexto debería liberar al desarrollador de este problema consiguiendo separar adecuadamente las distintas funcionalidades de la aplicación (separación de conceptos). Otra de las cuestiones a afrontar es la de establecer un método adecuado para modelar e integrar dicho contexto en el funcionamiento del sistema.

La sección siguiente recoge la revisión bibliográfica en el campo del contexto llevada a cabo en la presente tesis.

5.2. Trabajo relacionado organizado por categorías

Desde sus inicios, el esfuerzo de investigación en el área de la *computación sensible al contexto* se ha ido enfocando hacia cuatro direcciones principales: (1) *aplicaciones sensibles al contexto*; (2) arquitecturas software para soportar la construcción de *aplicaciones sensibles al contexto*; (3) desarrollo de middlewares para el soporte de aplicaciones distribuidas *sensibles al contexto* (middlewares del contexto), los cuales ofrecen en este caso servicios relacionados con el contexto; y (4) modelado del contexto.

Las primeras *aplicaciones sensibles al contexto* fueron diseñadas como aplicaciones autónomas, cada una manejando su propia información contextual. Este enfoque limita la reutilización de componentes e impide la posibilidad de compartir el contexto.

Posteriormente, las arquitecturas del contexto introducen componentes genéricas que pueden ser reutilizadas. Aún así, la composición de componentes está fuertemente acoplada a la aplicación y al entorno particular, haciendo difícil construir *aplicaciones sensibles al contexto* que puedan adaptarse automáticamente a distintos entornos.

Para aliviar este problema se introducen los middlewares del contexto, que consiguen desacoplar la aplicación de las fuentes del contexto, haciendo posible compartirlo entre aplicaciones, permitiendo que éstas trabajen sin restricciones en diversos entornos. Los middlewares son sistemas distribuidos típicamente basados en servidor, por lo que se requiere un acceso permanente al mismo con los consecuentes inconvenientes en redes inalámbricas.

En concreto, los middlewares están enfocados hacia el desarrollo de *computación ubicua* o *pervasiva*. Por supuesto, la heterogeneidad impone retos importantes tanto en el modelado del contexto como en el diseño de infraestructuras, tanto a nivel de hardware (combinación de dispositivos diversos y distintas tecnologías de red integrando todos estos dispositivos), de software (presencia de distintos sistemas operativos y aplicaciones que requieren interoperabilidad) como también a nivel arquitectural (disparidad de interconexiones de red). Esta disparidad exige el desarrollo de middlewares adaptables y

escalables que ofrezcan mecanismos de comunicación transparentes para el desarrollador de aplicaciones.

En general, la mayoría de los middlewares desarrollados, o bien están restringidos a un único dominio del contexto, como por ejemplo en [CC03], o bien la complejidad inherente para procesar la información relativa al contexto ofrecida por los mismos impide su despliegue en dispositivos de recursos limitados.

Otro aspecto a considerar es la evolución del contexto. Los modelos del contexto deberían ser suficientemente flexibles y genéricos para permitir la incorporación de nueva información acerca del contexto. Del mismo modo, los middlewares deberían ser capaces de manejar eficientemente nuevo contexto.

A continuación se presenta un breve análisis de cada una de estas categorías, introduciendo trabajos significativos pertenecientes a cada una de ellas.

5.2.1. Aplicaciones sensibles al contexto

Una vez revisada la literatura en lo que respecta a las distintas *aplicaciones sensibles al contexto* desarrolladas, se ha establecido la siguiente división en subdominios, la cual permite identificar el propósito pretendido en cada caso, destacando cuál es la propiedad específica de cada una de ellas. Su estudio proporciona una visión general acerca de qué es lo que los investigadores han pretendido alcanzar con su desarrollo.

5.2.1.1. Herramientas de oficina y de reunión

Las herramientas de soporte en entornos de oficina y en el desarrollo de reuniones constituyen un importante tópico de investigación en el campo de la *computación sensible al contexto*. La meta perseguida es la de mejorar las tareas de oficina incrementando su funcionalidad, como por ejemplo la de redirigir automáticamente llamadas de teléfono.

Ejemplo significativo

Un buen ejemplo de este tipo de aplicaciones es Active Badge System desarrollada en el Olivetti Research Lab [WHFG92]. Se trata de un sistema para localizar la posición de los usuarios en un entorno de oficina.

La infraestructura consiste en el uso de insignias provistas de transmisores de infrarrojos, los cuales proporcionan información acerca de la localización de los usuarios a un servicio de localización centralizado a través de una red de sensores.

Otras herramientas

Otros ejemplos son: el sistema Cyberdesk desarrollado en el Instituto de Tecnología de Georgia [Dey98], Office assistant del MIT Media Lab [YS00] y el sistema ParcTab desarrollado en el Xeroc PARC [WSA+95].

Tipo de contexto manejado

El tipo de contexto utilizado en este tipo de herramientas es mayoritariamente la localización (tanto actual como la que consta en un historial) y la identidad.

Soporte posterior a este tipo de aplicaciones

Fruto de la experiencia adquirida con el uso del sistema ParcTab aplicado a distintos objetivos dentro del campo de las aplicaciones de soporte en entornos de oficina, se desarrolla la denominada arquitectura del sistema de Schilit [Sch95], presentada más adelante dentro de la categoría de middlewares (véase *sección 5.2.4.1.*).

5.2.1.2. Guías turísticas

La meta de las guías (generalmente guías turísticas) es la de proporcionar a los visitantes información sensible a la localización.

Ejemplo significativo

Un buen ejemplo de este tipo de aplicaciones es el proyecto GUIDE[1] de la Universidad de Lancaster [DMCB98], que desarrolló una guía turística para asistir a los visitantes de la ciudad de Lancaster.

Los usuarios se desplazan con un ordenador portátil que se comunica a través de una red de área local inalámbrica para recuperar información.

Basado en la localización del usuario y sus preferencias, la información es transmitida desde las estaciones base a los portátiles como parte de una planificación regular (información relevante de la región), o bien en respuesta a las peticiones de los clientes.

El sistema puede calcular la localización del usuario automáticamente, aunque también el usuario puede especificar dónde se encuentra en un momento dado.

Otras herramientas

Otros ejemplos son: el proyecto Cyberguide desarrollado en el Instituto de Tecnología de Georgia [AAH+97]. El enfoque que utiliza es totalmente distinto al de GUIDE. Se trata de un sistema portátil autónomo que dispone de toda la información que requiere preinstalada en el propio sistema. Cyberguide recibe un identificador de localización que

[1] http://www.guide.lancs.ac.uk

se transmite a través de emisores. Sólo tiene que utilizar ese identificador para encontrar localmente la información almacenada relativa a esa localización.

Tipo de contexto manejado

El conocimiento del usuario, sus preferencias y el conocimiento del entorno, incluyendo la localización física del usuario.

Soporte posterior a este tipo de aplicaciones

La proliferación de las guías turísticas móviles ha dado lugar a mucho trabajo de investigación en el campo. Como soporte de ayuda a la generación de este tipo de aplicaciones surgió la arquitectura Virtual Information Towers [LKRF99], presentada más adelante dentro de la categoría de middlewares (véase *sección 5.2.4.2.*).

5.2.1.3. Asistentes de campo

La meta de un asistente de campo (del término inglés *fieldwork*) es la de facilitar el trabajo de campo, generalmente en espacios abiertos, tratando de reducir la cantidad de información que el usuario –generalmente un ecologista- tiene que registrar. Así, cierto tipo de información como son la localización y el tiempo es percibido y adjuntado automáticamente a las observaciones y anotaciones realizadas por el usuario, no requiriendo su introducción expresa y consiguiendo por tanto una reducción de la interacción con el dispositivo.

Ejemplo significativo

La herramienta de asistencia de campo *sensible al contexto* desarrollada por la Universidad de Kent, descrita en [RPM97] y posteriormente en [PRM99] constituye el ejemplo por excelencia. Se construyó para asistir a un ecologista en la introducción de observaciones realizadas en el estudio de campo de las jirafas en una reserva de Kenia.

Otras herramientas

No existen otros grupos de investigación que se encuentren desarrollando este tipo de herramientas, aunque resulta ser un dominio de aplicación interesante.

Tipo de contexto manejado

El tipo de contexto manejado por estas herramientas es del tipo localización y tiempo.

Soporte posterior a este tipo de aplicaciones

La herramienta mencionada fue testeada en Kenia entre 1997 y 1998, llegando a la conclusión de que resultaba muy valiosa. En consecuencia, el grupo de investigación de la

Universidad de Kent ha reconocido la necesidad de una arquitectura que proporcione un acceso común al contexto habitual de este tipo de sistemas, lo que dio lugar al desarrollo de la arquitectura del contexto *Contextual Information Service* [Pas98] presentada más adelante dentro de la categoría de arquitecturas del contexto (véase *sección 5.2.3.1.*).

5.2.1.4. Prótesis de memoria

La meta de este tipo de herramientas es la de utilizar información contextual para ayudar a recordar cosas, generalmente asociadas a situaciones contextuales futuras (e.g. recordar la realización de ciertas tareas cuando el usuario se encuentra próximo al lugar donde puede llevarlas a cabo).

Este tópico se implantó como un dominio de aplicación importante dentro del campo de la computación sensible al contexto e introdujo la noción de memoria humana aumentada, conocida con el término de *prótesis de memoria*.

Este tipo de herramientas también son conocidas por el término inglés *remembrance tools*.

Ejemplo significativo

El trabajo clásico dentro de este dominio es Forget-me-not [LF94] desarrollado en el centro de investigación de Xerox, cuyo propósito fue ayudar a recordar eventos.

Otras herramientas

Otros ejemplos son: MemoClip [Bei00], desarrollado en la Universidad de Karlsruhe, capaz de recordar al usuario la realización de tareas basándose en su localización y CyberMinder [DA00a], desarrollado en el Instituto de Tecnología de Georgia. Este último es un sistema capaz de advertir tanto de eventos simples como de situaciones complejas.

Por último, pueden mencionarse también el asistente de la compra de los laboratorios A T&T Bell [ACK94] y el agente de prótesis de memoria del MIT Media Lab [SMR+97].

Tipo de contexto manejado

Algunas de ellas únicamente manejan la localización (MemoClip). Otras manejan también otro tipo de contexto materializado a través de eventos (CyberMinder y Forget-me-not).

Soporte posterior a este tipo de aplicaciones

Posteriormente se desarrolla un marco general para este tipo de herramientas tratando de ofrecer el equivalente electrónico al concepto de la vida cotidiana 'Post-it'. Se trata de la arquitectura del contexto *Stick-e notes* [Pas97] presentada más adelante dentro de la categoría de arquitecturas del contexto (véase *sección 5.2.3.1.*).

5.2.1.5. Herramientas de interfaz

Un último grupo de *aplicaciones sensibles al contexto* es el de las herramientas de interfaz. Su objetivo es el de ajustar automáticamente la interfaz de un dispositivo de acuerdo a las percepciones obtenidas por un conjunto de sensores integrados en los dispositivos móviles, tratando de proporcionar nuevas modalidades de interacción.

Ejemplo significativo

La investigación llevada a cabo en esta categoría de aplicaciones está protagonizada por el centro de investigación de Microsoft [HPSH00], denominado *Sensing on mobile devices*, donde se experimentan distintas formas de incrementar tanto la interacción como la funcionalidad trabajando con PDAs (PDAs aumentadas).

Otras herramientas

Se puede citar el proyecto Nomadic Radio del MIT Media Lab [SS00].

Tipo de contexto manejado

El contexto manejado es todo aquel que pueda ser percibido a través de los sensores integrados en los dispositivos con los que se experimenta.

Soporte posterior a este tipo de aplicaciones

El proyecto *Technology for Enabling Awareness* (TEA) [SAT+99] desarrollado por la Universidad de Karlsruhe tiene como objetivo servir de soporte en el desarrollo de nuevos conceptos de interacción en dispositivos móviles de uso personal. Este proyecto se presenta más adelante dentro de la categoría de arquitecturas del contexto (véase *sección 5.2.3.3.*).

5.2.1.6. Conclusiones acerca de las aplicaciones sensibles al contexto

Se han presentado cinco categorías de *aplicaciones sensibles al contexto*. En todas ellas se utiliza la información contextual para mejorar la operativa del sistema de muy diversas formas (añadiendo nueva funcionalidad al sistema, proporcionando información dependiente del contexto, reduciendo la interacción requerida con el dispositivo, proporcionando nuevas modalidades de interacción, o creando aplicaciones totalmente nuevas).

El análisis de las *aplicaciones sensibles al contexto* pone en evidencia la falta de un soporte para adquirir un rango más amplio de contexto a partir de una mayor variedad de sensores. El tipo de información contextual por excelencia es la localización. Otros tipos de contexto, utilizados de forma más puntual son el tiempo y la identidad. Por otro lado, los aspectos de personalización relativos al usuario tan sólo se consideran en el caso de las guías turísticas.

En consecuencia, el reto no está únicamente en construir herramientas del contexto, sino también en explorar la manera como utilizar nueva información contextual. Un tipo de contexto prometedor es el de la 'actividad' de una persona (e. g. dormir, comer, ver la televisión). Así por ejemplo, si una aplicación conoce en qué actividades está involucrado un individuo, puede determinar en base a su perfil de usuario, si éste desea o no ser interrumpido por algún tipo de comunicación.

El motivo por el que las aplicaciones no han podido hacer un uso más intensivo del contexto se debe, en parte, a la falta de un soporte uniforme para la construcción y ejecución de este tipo de aplicaciones. De hecho, la mayoría de *aplicaciones sensibles al contexto* han sido construidas ad hoc y de manera improvisada, fuertemente influenciadas por la tecnología subyacente utilizada para adquirir el contexto [Nel98]. En consecuencia, las aplicaciones no separan la adquisición del contexto del uso que se hace del mismo (bajo nivel de separación de conceptos), de manera que los desarrolladores de las aplicaciones no tienen otra alternativa que tratar directamente los detalles de la adquisición del contexto. Todo ello da lugar a una tendencia generalizada hacia el desarrollo de aplicaciones fuertemente acopladas que impiden su evolución debido a que la semántica de la aplicación no está separada de los detalles de los sensores.

Incluso cuando se utilizan abstracciones para el manejo de sensores, con el propósito de desacoplar su uso, no existe un soporte generalizado que permita extender estas abstracciones para satisfacer las necesidades y la problemática general de la *computación sensible al contexto*. Esta falta de generalidad va en detrimento de la reutilización, resultando casi imposible integrar soluciones ya existentes a aplicaciones que no son *sensibles al contexto*, o incorporar nuevo contexto a las que ya lo son.

Otro aspecto igualmente relevante es que en muchas ocasiones las aplicaciones no soportan la notificación de cambios en el contexto, requiriendo que la consulta y su posterior análisis sean resueltos por la propia aplicación, con objeto de determinar si se ha producido un cambio relevante.

Con el propósito de desarrollar un marco general para facilitar la construcción y evolución de las aplicaciones de este tipo surgen las primeras arquitecturas del contexto y posteriormente los middlewares del contexto, como abstracciones que soportan su complejo desarrollo.

En la sección siguiente se relacionan los requisitos que debería cumplir cualquier herramienta de este tipo, utilizándose como elemento comparativo en el análisis realizado a continuación.

5.2.2. Requisitos de una infraestructura de soporte al contexto

Dey define los siete siguientes aspectos que es deseable que cumpla una infraestructura de soporte al contexto, los cuales constituyen la base para el desarrollo de su herramienta Context Toolkit [Dey00]. Entre ellos se incluyen también las consideraciones de red para aplicaciones distribuidas sobre plataformas heterogéneas, destinadas a entornos ubicuos y pervasivos. Son los siguientes:

1. Separación de conceptos y manejo del contexto (interfaz común)

La información contextual utilizada por las aplicaciones podría provenir directamente de los sensores o por el contrario podría pasar a través de varias capas de abstracción. Un diseñador de aplicación no necesita conocer los pasos requeridos para interpretar la información contextual, sino que el proceso de adquisición debería serle transparente. Por lo tanto, todas los componentes necesitan tener una interfaz externa común, de manera que una aplicación pueda tratar esa información uniformemente.

2. Especificación del contexto (suscripción / mecanismo de sondeo)

Debe haber un mecanismo que permita a una aplicación indicar en qué información contextual está interesada y cuándo le interesa ser notificada. Una aplicación debe ser capaz de solicitar esta información a la arquitectura del contexto, o bien de subscribirse a ciertos servicios de información contextual.

3. Interpretación del contexto

Este requisito hace referencia a la transformación de uno o más tipos de contexto en otro tipo de contexto resultante de la consideración de todos ellos en conjunto.

Muchas aplicaciones comúnmente llevan a cabo una interpretación similar de la información contextual (e. g. transformar un número de identificación en una dirección de correo electrónico). En ese sentido, es deseable proporcionar técnicas reutilizables para interpretar la información contextual.

4. Disponibilidad constante para la adquisición del contexto

Los componentes que adquieren el contexto deben ejecutarse independientemente de las aplicaciones que hacen uso del mismo, de manera que la información contextual esté siempre disponible para diferentes aplicaciones. Una ventaja adicional es que el diseñador de la aplicación no necesita instanciar, mantener o hacer un seguimiento de los componentes que adquieren el contexto.

5. Almacenamiento del contexto

El historial del contexto debe ser almacenado de tal manera que las aplicaciones y las arquitecturas del contexto puedan utilizarla para establecer tendencias y predecir futuros valores del contexto.

Los dos últimos requisitos son específicos para el desarrollo de middlewares del contexto.

6. Descubrimiento de recursos

Es deseable la intervención de un mecanismo responsable de detectar nuevos recursos, servicios o dispositivos, así como de proveer a la aplicación de mecanismos para acceder a ellos.

7. Transparencia en el mecanismo de comunicación

El hecho de que la comunicación sea distribuida debería ser transparente tanto para los sensores como para las aplicaciones. Este aspecto libera al diseñador de tener que construir un framework de comunicación.

5.2.3. Arquitecturas del contexto

Antes de entrar en detalle en la revisión y comparativa de las distintas arquitecturas del contexto seleccionadas, es conveniente definir el término *"arquitectura del contexto"*.

Una *arquitectura del contexto* es una arquitectura software que maneja información contextual y soporta el desarrollo de *aplicaciones sensibles al contexto* [Ens02].

Por otro lado, Boasson define una arquitectura software como

> *"una estructura del sistema que consiste en un conjunto de módulos de software activos, un mecanismo de interacción entre los módulos y un conjunto de reglas que gobiernan la interacción"[Boa95].*

Una arquitectura del contexto se compone por tanto de módulos software que soportan el desarrollo de una *aplicación sensible al contexto*, de manera que facilita la integración de la información contextual, así como su entrega a las aplicaciones.

La figura 5.1 representa una concepción simplificada de un posible modelo de arquitectura del contexto. Las flechas indican el sentido de la interacción entre los componentes. Los interrogantes indican que el flujo de información señalado por las flechas no siempre se aplica, quedando supeditado a cada arquitectura del contexto en particular.

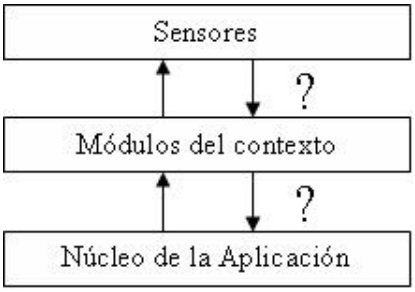


Figura 5.1: Un modelo simplificado de arquitectura del contexto.

Es deseable que las arquitecturas del contexto proporcionen la habilidad para reutilizar sensores o permitir la evolución de las aplicaciones existentes, de manera que puedan trabajar con distintos sensores y tipos de contexto.

Las arquitecturas del contexto que se presentan a continuación son las resultantes del esfuerzo de abstracción llevado a cabo en cada uno de los dominios de aplicación en el campo de la *sensibilidad al contexto* descritos en la *sección 5.2.1*, tal y como allí se han ido mencionando.

5.2.3.1. Contextual Information Service

Contextual Information Service (CIS) es una propuesta de arquitectura para soportar *aplicaciones sensibles al contexto* del tipo asistentes de campo (véase *sección 5.2.1.3.*), la cual ha sido desarrollada por la Universidad de Kent [Pas98]. Su meta es mantener un modelo contextual que facilite la obtención de datos contextuales habituales en este tipo de sistemas.

Un ejemplo de aplicación construida utilizando CIS es un asistente de campo vestible (del término inglés 'weareable') diseñado para asistir la recogida de observaciones de una jirafa por parte de un ecologista, similar aunque más completa que la aplicación mencionada en la *sección 5.2.1.3.* La aplicación automáticamente percibe la localización del ecologista de manera que éste no necesita tener que introducir información de ese tipo.

Desafortunadamente, existe muy poca información disponible sobre CIS y no es posible descargarla.

5.2.3.1.1. Arquitectura

Este modelo consiste en un mundo de 'artefactos', esto es, objetos que tienen un nombre, tipo y conjunto de 'estados' contextuales. Por ejemplo, el artefacto 'persona'

tiene un estado correspondiente a su 'localización'. Es posible añadir nuevos artefactos seleccionando una plantilla de un catálogo de artefactos.

La arquitectura contiene un diálogo de estado que lista varios estados genéricos que pueden ser utilizados para construir artefactos. Cada estado tiene uno o más formatos y tipos, métodos para realizar traducciones entre ellos, así como un conjunto de operaciones. Los clientes (aplicaciones) CIS pueden acceder a cualquier estado de cualquier artefacto. La figura 5.2 ilustra el modelo CIS.

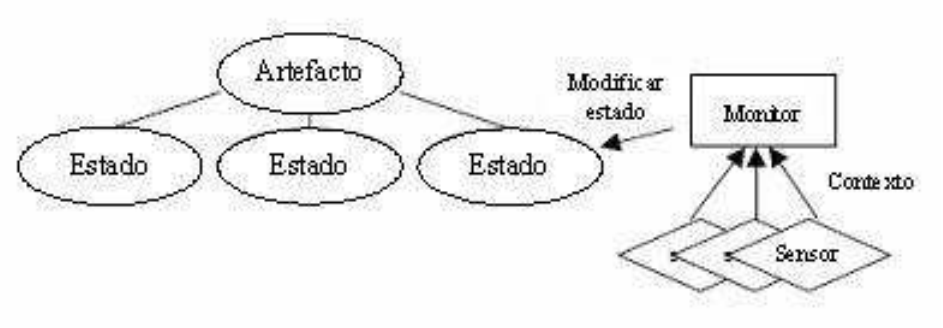


Figura 5.2: Modelo CIS.

Los llamados 'monitores' se asocian con cada estado del artefacto y utilizan directivas de 'calidad de servicio' para elegir los sensores que necesitan para obtener información contextual. Los sensores son programas que extraen información contextual de dispositivos conectados o embebidos. El monitor decide cuándo obtener o generar valores, o bien ejecutar simulaciones.

5.2.3.1.2. Análisis de la satisfacción de requisitos

Esta arquitectura tiene ciertas características interesantes como son el hecho de soportar la interpretación del contexto y la elección de un sensor basada en una garantía de calidad de servicio. Sin embargo, la conexión entre las aplicaciones y los sensores subyacentes es fuertemente acoplada, dando lugar a un enfoque dependiente de la aplicación para la construcción del sistema.

CIS mantiene un modelo orientado a objetos que tiene un conjunto de estados predefinidos. Los objetos pueden ser enlazados entre sí a través de relaciones del tipo "cerca de". Por ejemplo, el conjunto de árboles cercanos serían especificados utilizando este tipo de relación con un usuario. Esa información debe ser modificada dinámicamente conforme el usuario se desplaza en un determinado entorno.

No existe ninguna indicación acerca de cómo implementar estas características ni tampoco proporciona un soporte explícito para añadir nuevos sensores y tipos de contexto. Otras características inexistentes son las siguientes: comunicación distribuida, almacenamiento del contexto, descubrimiento de recursos o disponibilidad constante de componentes de gestión del contexto. Un último detalle es que CIS dispone de un modelo del entorno.

5.2.3.2. Stick-e Notes

La arquitectura Stick-e notes[2] [Pas97] es un marco general para soportar el desarrollo de *aplicaciones sensibles al contexto* no distribuidas pertenecientes a la categoría de *prótesis de memoria* (véase *sección 5.2.1.4.*), aunque también se aplica para el trabajo de campo y el desarrollo de guías turísticas. Ha sido desarrollada en la Universidad de Kent.

La meta de este trabajo es la de permitir la construcción de servicios *sensibles al contexto* sin necesidad de ser programador. Proporciona un mecanismo general para especificar qué contexto desea ser utilizado por un diseñador de aplicación y una semántica para escribir las reglas que especifican las acciones a tomar ante combinaciónes de contexto.

Stick-e note constituye una metáfora del concepto de 'Post-it' que se compone de dos partes: una nota que especifica el contexto que debe ser cumplido y un cuerpo que especifica qué acción debe ser tomada cuando el contexto de la nota es satisfecho. El conjunto de etiquetas que pueden ser utilizadas viene establecido a través de una especificación DTD.

Un ejemplo de aplicación generada con Stick-e notes es un servicio de recordatorio para recordar al usuario una cita inminente cuando encuentra a una persona. Otro ejemplo es la construcción de una aplicación de guía turística. Un autor podría escribir reglas individuales a través del uso de stick-e notes. La figura 5.3 muestra un ejemplo:

```
<note>
<required>
<at> (1,4)..(3,5)
<facing> 150..210
<during> december
<body>
The large floodlit building
At the bottom of the hill
Is the cathedral.
```

Figura 5.3: Ejemplo de una nota utilizada por el sistema Stick-e notes.

[2]Disponible gratuitamente en http://www.cs.ukc.ac.uk/projects/mobicomp/Fieldwork/Software/.

Esta nota representa la regla siguiente: *"cuando el usuario está localizado entre las coordenadas (1,4) y (3,5), y orientado entre 150 y 210 grados durante el mes de diciembre, proporciona información acerca de la catedral"*.

5.2.3.2.1. Arquitectura

La arquitectura de stick-e note consiste en tres tipos de componentes:

- *Componente de actuación (SeTrigger)*: componente que contrasta el contexto presente del usuario con el contexto cargado en la stick-e note. Cuando ambos coinciden la nota es disparada.
- *Componente de ejcución (SeShow)*: componente de ejecución a la que se pasan las notas disparadas con objeto de ejecutar el cuerpo de la nota.
- *Componentes sensores (SeSensor)*: conjunto de componentes sensor que periódicamente alimentan la información manejada por el módulo *SeTrigger*. Estas componentes pueden estar distribuidas a lo largo de una red.

5.2.3.2.2. Análisis de la satisfacción de requisitos

El enfoque, aunque interesante, está muy restringido por el hecho de haber tomado la decisión de soportar autores no programadores. La semántica para escribir reglas es limitada y no puede manejar datos discretos frecuentemente cambiantes. Proporciona un mecanismo central para determinar cuándo las cláusulas de una regla dada se producen, limitando la escalabilidad.

Se ocultan los detalles de adquisición del contexto, sin embargo no existe un soporte para solicitar, almacenar e interpretar el contexto.

5.2.3.3. Technology for Enabling Awareness

El proyecto Technology for Enabling Awareness (TEA) consiste en un controlador de sensores desarrollado por la Universidad de Karlsruhe [SAT+99]. Su objetivo es el de desarrollar nuevos conceptos de interacción en dispositivos móviles de uso personal, por lo que puede catalogarse como un soporte para el desarrollo de *aplicaciones sensibles al contexto* dentro de la categoría de herramientas de interfaz (véase *sección 5.2.1.5.*).

5.2.3.3.1. Arquitectura

Utiliza un modelo de pizarra que consta de los siguientes componentes:

- *cues*. Es así como se llama a las representaciones de los sensores. Son capaces de escribir el contexto adquirido.
- *fact abstractors*. Es así como se llama a los interpretadores del contexto. Se encargan de leer y escribir el contexto interpretado o abstraído. Las aplicaciones leen el contexto relevante de estos componentes.

Esta arquitectura se utiliza para determinar el estado de un teléfono móvil con el objetivo de establecer automáticamente su perfil. Por ejemplo, si el usuario está caminando en un entorno exterior, el teléfono se pone en un modo particular de manera que el volumen se pone al máximo y el vibrador es activado. En cambio, si se supone que está en una reunión, se pone en estado de silencio y todas las llamadas se dirigen a un buzón de voz.

5.2.3.3.2. Análisis de la satisfacción de requisitos

Los componentes *cue* y *fact abstractor* proporcionan una separación de conceptos que permite separar la adquisición del contexto de su uso por parte de las aplicaciones.

Proporciona soporte limitado para especificar en qué contexto está interesado una aplicación. Además, no proporciona soporte para una constante disponibilidad ni tampoco para almacenar el contexto.

5.2.4. Middlewares del contexto

Desde la aparición de las primeras arquitecturas del contexto existe un esfuerzo común hacia la incorporación de sensores del contexto y proveedores de diferentes tecnologías, considerándose como una evolución del sistema en su conjunto [dRE05].

Si bien las arquitecturas se enmarcan en el desarrollo de aplicaciones que no necesariamente son distribuidas, para facilitar el desarrollo de este otro tipo de entornos *sensibles al contexto* se debe recurrir al uso de un middleware del contexto.

Un *middleware* es un software de conectividad que ofrece un conjunto de servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas. Funciona como una capa de abstracción de software distribuida, que se sitúa entre las capas de aplicación y las capas de red subyacentes (ignorando por tanto aspectos

como la heterogeneidad de plataformas, el sistema operativo y los protocolos de comunicación). Su misión es la de desacoplar las aplicaciones de cualquier dependencia con las capas subyacentes, proporcionando una API para facilitar la programación y manejo de aplicaciones distribuidas [Bir04].

En efecto, la heterogeneidad genera escenarios muy complejos para resolver el acceso al contexto, dado que éste tiende a estar altamente distribuido imponiendo restricciones de comunicación. El hecho de que la comunicación es distribuida debería ser transparente tanto para los sensores como para las aplicaciones, liberando al diseñador de tener que construir un framework de comunicación. Este aspecto se corresponde con el requisito de soporte al contexto número 7 identificado por Dey (véase *sección 5.2.2.*), específico para plataformas heterogéneas.

Por último, para soportar la posible evolución del contexto, algunos middlewares exploran el concepto de descubrimiento de recursos, como por ejemplo Virtual Information Towers [LKRF99] y Context Toolkit [Dey00]. Este soporte permite descubrir y usar nuevos tipos de contexto y fuentes de información a lo largo de la ejecución. De hecho, esta funcionalidad se corresponde con el requisito de soporte al contexto número 6 identificado por Dey (véase *sección 5.2.2.*), específico para plataformas heterogéneas.

Uno de los trabajos más destacados en el desarrollo de middlewares del contexto es Context Toolkit [Dey00], que ofrece la abstracción denominada *widget del contexto*, aplicada a proveedores e interpretadores del contexto.

Los middlewares que se presentan a continuación han sido seleccionadas o bien por su relevancia y la influencia que han ejercido en el campo desde su aparición (es el caso de Context Toolkit), o bien –al igual que en el caso de las arquitecturas del contexto- por tratarse del resultado del esfuerzo de abstracción llevado a cabo en cada uno de los dominios de aplicación descritos en la *sección 5.2.1*, tal y como allí se han ido mencionando. Es el caso de la arquitectura del sistema de Schilit y de Virtual Information Towers.

5.2.4.1. La arquitectura del sistema de Schilit

En 1995 Schilit presentó en su tesis una arquitectura que soporta computación móvil *sensible al contexto* [Sch95], constituyendo el primer trabajo en este campo y que ha servido de inspiración a muchos otros grupos de investigación que trabajan en este tópico.

La tesis de Schilit se basa en la experiencia adquirida de numerosos experimentos realizados con el sistema ParcTab[3] en el campo de la gestión. El sistema ParcTab había sido desarrollado en el centro de investigación de Xerox PARC (Palo Alto) para explorar

[3]http://sandbox.xerox.com/parctab/

las capacidades y el impacto de los ordenares móviles en un entorno de oficina [WSA+95]. Por lo tanto este trabajo está centrado en el desarrollo de *aplicaciones sensibles al contexto* dentro de la categoría de aplicaciones de oficina (véase *sección 5.2.1.1.*).

El sistema ParcTab es una versión primitiva de PDA (ordenador móvil del tamaño de la palma de la mano), provista de conexión por infrarrojos que permite la comunicación entre ellos y con ordenadores de sobremesa a través de una red de área local. La figura 5.4 muestra una imagen de este dispositivo.

Figura 5.4: El sistema ParcTab.

Gracias a los sensores de infrarrojos situados en cada sala y al uso de *mapas activos*, el sistema es consciente de la localización de los distintos ParcTabs. Un *mapa activo* es un mapa del edificio que es continuamente actualizado. Por lo tanto, es capaz de mostrar qué ParcTabs se encuentran en la misma sala, notificando a las aplicaciones de los cambios en la localización.

Desafortunadamente, no es posible descargar el sistema ParcTab.

Las aplicaciones más populares de ParTab incluyen operaciones del tipo: lectura de correo electrónico, obtención de información del tiempo, exploración de ficheros, almacenamiento de información e introducción de notas como las más significativas.

5.2.4.1.1. Arquitectura

Se trata de una arquitectura basada en agentes que reúne información contextual acerca de los dispositivos y los usuarios. Se compone de tres componentes principales:

- Agentes de dispositivo que mantienen el estado y las capacidades de los dispositivos.

- Mapas activos que mantienen la información de la localización de los dispositivos y usuarios.

- Agentes de usuario que mantienen las preferencias de los usuarios.

Existe exactamente un agente de dispositivo por cada ParcTab que actúa como un conmutador para conectar las aplicaciones vía sensores/receptores de infrarrojos, y que se encarga de gestionar canales de comunicación con la aplicación.

El agente de usuario hace un seguimiento de la información personal del usuario, la cual puede incluir, entre otras cosas, sus preferencias, las características del dispositivo que pone en funcionamiento ese usuario, el tiempo y localización de la interacción más reciente. Toda esta información puede ser solicitada al agente de usuario desde las propias aplicaciones del usuario, o bien incluso desde aplicaciones ejecutadas por otros usuarios.

5.2.4.1.2. Análisis de la satisfacción de requisitos

Schilit y su grupo de investigación en Xerox PARC desarrollaron un conjunto de *aplicaciones sensibles al contexto* principalmente para aplicaciones móviles, las cuales están todas ellas centradas en la localización de los usuarios, con finalidades como la de localizar la impresora más cercana o mostrar mensajes en pantallas cercanas al usuario.

La arquitectura de Schilit soporta la recopilación de contexto acerca de dispositivos y usuarios, pero no proporciona ningún tipo de guía para su adquisición. Por el contrario, los agentes de usuario y dispositivo están construidos en una base individual, específica para el conjunto de sensores utilizados (sensores infrarrojos), lo que impide la evolución de las aplicaciones.

El requisito correspondiente a la separación de conceptos tan sólo se asume parcialmente. Cada vez que se requiere incorporar nuevos tipos de contexto es necesario reescribir los agentes del usuario y del dispositivo.

La arquitectura, no obstante, soporta la entrega del contexto a través de un sistema de consulta y unos mecanismos de notificación eficientes. También resuelve satisfactoriamente una disponibilidad constante para la adquisición del contexto.

La arquitectura soporta parcialmente la noción de descubrimiento, permitiendo a las aplicaciones encontrar los componentes en las que está interesada.

La interpretación del contexto no está soportada, requiriendo que sean las propias aplicaciones las que proporcionen un soporte propio.

Finalmente, es remarcable la ausencia de un soporte de almacenamiento del contexto.

5.2.4.2. Virtual Information Towers

La Universidad de Stuttgart ha desarrollado un sistema de información geográfica sensible a la localización que soporta el desarrollo de aplicaciones de la categoría de guías turísticas (véase *sección 5.2.1.2.*) denominado Virtual Information Towers (VITs) [LKRF99].

Su objetivo es el de facilitar el acceso y entrega de información sensible a la localización a dispositivos móviles. Sin embargo, existe muy poca información al respecto de este proyecto

5.2.4.2.1. Arquitectura

Se trata de un sistema descentralizado formado por un conjunto de nodos. Cada uno de estos nodos es un VIT.

Cada VIT contiene información específica de una cierta área geográfica, la cual puede consistir en una jerarquía de objetos de información denominados *posters* (páginas Web, audio, vídeo, etc.). El contenido de un VIT se sintetiza en un perfil que es el que se utiliza para llevar a cabo la búsqueda de información relativa a un determinado tópico.

La información contenida en un VIT puede dejar de ser relevante en un momento dado. Este problema se resuelve etiquetando dicha información con dos etiquetas de tiempo relativas al momento de la última modificación y el momento de expiración. Por otro lado, un VIT tiene un propietario con unos derechos de acceso.

La Universidad de Stuttgart ha desarrollado un prototipo que consiste en un Cliente VIT, de manera que a través de un navegador Web permite al usuario visualizar el contenido de un VIT, notificándole de cambios en el contexto.

Los VITs pueden encontrarse utilizando el Directorio de VITs global, un servicio distribuido comparable al Domain Name Server. Este directorio almacena por cada VIT el nombre, área de visibilidad, algunas palabras clave y la dirección.

El prototipo construido contiene también un gestor de VITs que almacena y gestiona (añade, borra o modifica) la información de un conjunto de VITs.

El único contexto manejado por este proyecto es la localización y el tiempo. Su antecesor fue el proyecto Nexus [FKV00]. El esfuerzo ha sido focalizado en el desarrollo de una librería Java para facilitar el acceso a la funcionalidad que el sistema de información geográfica proporciona. También se ha estudiado la manera de utilizar conjuntamente distintos modelos de localización.

5.2.4.2.2. Análisis de la satisfacción de requisitos

Este middleware resuelve el descubrimiento de recursos como requisito propio de un sistema distribuido, aunque no resuelve la transparencia en el mecanismo de comunicación.

Adicionalmente, resuelve el almacenamiento del contexto y la constante disponibilidad para su adquisición. La separación de conceptos está resuelta tan sólo parcialmente.

Por último, proporciona un modelo del entorno, así como también un soporte para la descripción del contexto.

5.2.4.3. Context Toolkit

Context Toolkit[4] es un proyecto del Instituto de Tecnología de Georgia [Dey00]. Consiste en una arquitectura del contexto que soporta el desarrollo y despliegue de *aplicaciones sensibles al contexto*, entendiendo por contexto cualquier información del entorno operativo de la aplicación que pueda ser percibido por ésta.

A grandes trazos, esta arquitectura consiste en los denominados *widgets del contexto* y una infraestructura distribuida que hospeda esos widgets. Los *widgets del contexto* son componentes software que proporcionan acceso a la información contextual a las aplicaciones ocultando los detalles acerca de cómo se lleva a cabo su adquisición, tal y como un *widget* de interfaz de usuario aísla a las aplicaciones de ciertos aspectos de la presentación.

5.2.4.3.1. Arquitectura

La arquitectura utiliza un enfoque orientado a objetos consistente en tres tipos de objetos: *widgets* del contexto, *interpretadores* del contexto y *agregadores* del contexto.

Widgets del contexto

Como ya se ha mencionado, los *widgets del contexto* proporcionan información contextual obtenida, por ejemplo, vía sensores.

El gran aporte respecto a la arquitectura de Schilit es el hecho de que estos widgets son módulos personalizables que abstraen los detalles acerca de la percepción y adquisición del contexto a los componentes que hacen uso de ellos. Son, por lo tanto, componentes software reutilizables.

Un *widget* del contexto consta de atributos y *callbacks*. Los atributos son las piezas de contexto que están disponibles para su utilización por otras componentes. Los *callbacks*

[4]Disponible gratuitamente en http://www.cc.gatech.edu/fce/contexttoolkit/.

son los tipos de eventos que el *widget* utiliza para notificar a los componentes que están subscritas al *widget*. Dichas componentes pueden hacer uso de ambos y a través de mecanismos de sondeo y notificación ser capaces de recuperar información contextual. Además, la información histórica acerca del contexto también puede ser recuperada.

Interpretadores del contexto

Los interpretadores del contexto son responsables de interpretar la información del contexto, siendo capaces de transformar diferentes tipos de contexto de manera que le resulte útil a la aplicación. Por ejemplo, un interpretador del contexto puede averiguar el nombre de una persona cuando su número identificativo es percibido por un sensor, o bien transformar un sistema de coordenadas a uno específico para la aplicación.

Agregadores del contexto

Agregan valores del contexto relacionados para obtener información contextual de más alto nivel. Por ejemplo, combinar la presión arterial, el peso y la temperatura de una persona para calcular su estado físico. Es decir, su misión es la de recoger el contexto completo acerca de una entidad, como en el caso de la persona.

Además, es la componente responsable de suscribir cada *widget* del contexto en el que se está interesado y actuar como un proxy para la aplicación.

Al igual que un *widget del contexto*, consta de atributos y *callbacks*. Puede ser subscrito y sondeado, y su historial puede ser recuperado.

La figura 5.5 representa la interacción típica entre componentes y aplicaciones en Context Toolkit.

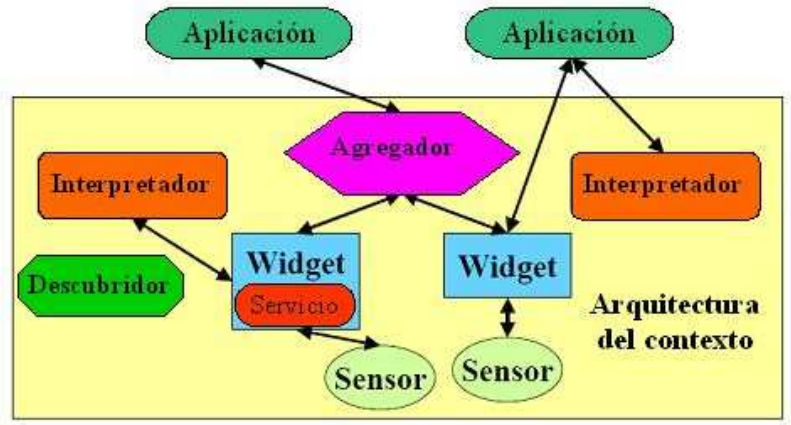


Figura 5.5: Arquitectura de Context Toolkit.

5.2.4.3.2. Análisis de la satisfacción de requisitos

De manera resumida, los servicios que ofrece Context Toolkit son los siguientes:

- encapsulamiento de los sensores
- acceso a datos del contexto a través de una API de red
- abstracción de los datos del contexto a través de interpretadores
- compartición de datos del contexto a través de una infraestructura distribuida
- almacenamiento de los datos del contexto a través de un historial
- control de acceso básico para protección de la privacidad

Esta arquitectura contiene también una componente de descubrimiento que actúa como un registro de todos los componentes y sus capacidades, acción realizada de forma automática. Las aplicaciones pueden solicitar y suscribir en el descubridor los componentes en las que están interesadas. También pueden solicitar la realización de interpretaciones por parte de interpretadores individuales.

Los componentes (*widgets*, *interpretadores* y *agregadores*) son instanciados independientemente de los demás, pudiendo ejecutarse en diferentes dispositivos.

Todos los componentes y aplicaciones utilizan instancias de *BaseObjet* (una clase de Java que se encarga de la comunicación distribuida) para comunicarse con las otras componentes. De ese modo, no es necesario conocer la manera como se implementa la comunicación con las otras componentes.

Tanto los *agregadores* como los *widgets del contexto* manejan un registro (*log*) de subscripción que permite reestablecer la comunicación entre los componentes y las aplicaciones que han sido subscritas si el sistema cae.

La implementación en Java de Context Toolkit utiliza el protocolo de comunicación HTTP y XML como formato de mensaje. Sin embargo, es sencillo cambiar este sistema de comunicación.

Los *widgets del contexto* y los *agregadores* automáticamente almacenan el contexto que adquieren.

5.2.5. Modelos del contexto

Por lo que respecta a los modelos del contexto, de acuerdo a Strang et al. en [SLP04], los únicos modelos apropiados para modelar el contexto dirigida a entornos de computación pervasiva son los modelos basados en orientación a objetos y los basados en ontologías.

Existen varios proyectos que han adoptado el paradigma orientado a objetos para modelar información acerca del contexto, como por ejemplo GUIDE [DMCB98]. Se trata de un enfoque simple, eficiente y fácil de desplegar, teniendo en cuenta su amplia difusión. Sin embargo, el uso de un modelado del contexto orientado a objetos requiere la implementación de infraestructuras que soporten todas las operaciones del contexto relacionadas con su almacenamiento y adquisición.

Actualmente la mayoría de investigación en modelado del contexto está dirigida hacia el uso de ontologías para especificar información contextual y las relaciones entre contextos [CFJ03], [RC04], [SLF03]. El uso de ontologías proporciona un paradigma poderoso para el modelado del contexto con un alto grado de expresividad, así como también el soporte adecuado a la heterogeneidad. Sin embargo, para procesar ontologías la arquitectura del sistema debe proporcionar un motor de ontología capaz de procesar inferencias. Este requisito puede impactar en el manejo del contexto a nivel local en el caso de dispositivos de recursos limitados, al imponer la necesidad de mantener ese motor de ontología en el servidor.

Generalmente, los middlewares que adoptan un modelado del contexto basado en ontologías [RC04], [HKK04] adoptan un paradigma de interacción basado en agentes, como por ejemplo en [CFJ03], posibilitando el procesamiento de *computación sensible al contexto* también en dispositivos de recursos limitados. Sin embargo, este enfoque incrementa considerablemente el número de interacciones necesarias, generalmente a través de enlaces inalámbricos, limitando el uso del contexto cuando el dispositivo trabaja sin conexión.

Otra posibilidad para mejorar este aspecto es la adopción de paradigmas de comunicación específicos para sistemas móviles, los cuales son más apropiados para el uso de redes inalámbricas, teniendo en cuenta las posibilidades de intermitencia en la conectividad. Se está haciendo referencia a los enfoques de comunicación conocidos como *publish/subscribe* [SER+04] y *tuplespaces-based* [HL01], que ya han sido adoptados por varios middlewares.

5.3. Comparativa entre infraestructuras del contexto

A continuación se sintetiza las observaciones derivadas de la revisión de la literatura en el campo de herramientas del contexto. En primer lugar se ofrece una comparativa

de las infraestructuras analizadas en este capítulo, y a continuación se incluyen otras observaciones acerca de otras herramientas no incluidas en este análisis.

5.3.1. Comparativa entre las infraestructuras del contexto analizadas

La tabla 5.6 indica en qué medida las infraestructuras descritas en las secciones previas cumplen los requisitos considerados, descritos en la *sección 5.2.2*.

√ → Cubierto. ½ → Parcialmente cubierto. — → No cubierto.		REQUISITOS						
		Contexto					*Distrib.*	
		Separación de Conceptos (Interfaz común)	Especificación del contexto (Suscripción/Sondeo)	Interpretación del contexto	Disponibilidad constante adquisición del contexto	Almacenamiento del contexto	Descubrimiento de recursos	Transparencia mecanismo comunicación
ARQUITECTURA	Contextual Information Service (CIS)	½	½	√	—	—	—	—
	Stick-e Notes	√	½	—	½	—	½	—
	Technology for Enabling Awareness (TEA)	√	½	√	—	—	½	—
MIDDLEWARE	Arquitectura del sistema de Schilit	½	√	—	√	—	½	½
	Virtual Information Towers (VITs)	½	—	—	√	√	√	—
	Context Toolkit	√	√	√	√	√	√	√

Figura 5.6: Tabla comparativa de las infraestructuras del contexto presentadas.

Como puede observarse, ninguna de ellas soporta todas las características necesarias para soportar aplicaciones sensibles al contexto de manera sistemática, a excepción de Context Toolkit. Entre algunas de sus ventajas, el hecho de que el modo de comunicación entre los objetos pueda ser modificado constituye una propiedad de gran utilidad, posibilitando la incorporación de esta arquitectura en otros middlewares.

El tipo de contexto manejado en general es limitado, echándose en falta la exploración de técnicas que intenten combinarlos explícitamente con los aspectos relativos al usuario, tal y como se indica en el *Capítulo 1* (véase *Capítulo 1*; *sección 1.2*). Este aspecto tan sólo se contempla en la arquitectura del sistema de Schilit.

Por otro lado, a pesar de la relevancia que supone disponer de detalles de implementación de las distintas infraestructuras, a fin de utilizarlas como base de estudio, en general no existe suficiente información disponible acerca de los distintos proyectos de la que poder extraer conclusiones e identificar nuevos tópicos. Context Toolkit constituye una notable excepción.

En la categoría de arquitecturas del contexto analizadas existen muchos aspectos a mejorar, entre ellos el más desfavorecido es el del almacenamiento del contexto. La ausencia de este tipo de soporte imposibilita la adquisición de información contextual previa por parte de las aplicaciones, limitando el volumen de análisis del contexto que puede ser llevado a cabo por la aplicación, lo que constituye una parte integral de la actividad de una *aplicación sensible al contexto*.

Por otro lado, en los enfoques basados en arquitectura los mecanismos internos que soportan la auto-modificación suelen ser muy específicos del dominio de aplicación y el acoplamiento de la arquitectura con la aplicación es generalmente considerable.

En la categoría de middlewares la asignatura pendiente es la interpretación del contexto, mientras que el requisito número 4 (disponibilidad constante en la adquisición del contexto) es el que obtiene mejor puntuación, por supuesto haciendo siempre referencia al conjunto de middlewares analizados.

5.3.2. Observaciones acerca de otras infraestructuras no incluidas en el análisis

Al margen de las herramientas analizadas aquí, las tecnologías dominantes en el campo de los middlewares del contexto son las de sistemas basados en agentes (utilizada en la arquitectura del sistema de Schilit) y también las técnicas de reflexión (no aplicadas en ninguna de las herramientas analizadas).

En particular, podría ser interesante investigar en qué medida un sistema de agentes resulta idóneo en este campo, sobretodo teniendo en cuenta la problemática de su despliegue a través de redes inalámbricas.

Por otro lado, resulta imprescindible mencionar también las técnicas de reflexión y las arquitecturas denominadas meta-nivel, las cuales son capaces de adaptarse a requisitos cambiantes. La literatura muestra que un enfoque de este tipo es apropiado por su capacidad para llevar a cabo auto-modificaciones en el comportamiento existente. Ejemplos que avalan esta teoría son [CEM03], [MB02], [PL97], [LSZB98] y la infraestructura Contextor de [RC04]. En efecto, las capacidades de introspección (capacidad de describir su propio comportamiento) y reflexión (capacidad para analizar su propio estado y auto-adaptarse), juntamente con las propiedades de separación de conceptos, transparencia y flexibilidad asociadas hacen que este enfoque sea comúnmente reconocido como un mecanismo valioso para el desarrollo, extensión y mantenimiento de software, así como para el desarrollo de middlewares [CCL00].

En concreto, existe un tipo particular de arquitectura reflexiva denominada arquitectura *Adaptive Object-Model* (AOM) [YJ02]. Se trata de un sistema que representa clases, atributos, relaciones y comportamiento como *metadatos*. Estos sistemas almacenan su modelo de objetos en ficheros XML o en bases de datos, con objeto de ser interpretados en tiempo de ejecución y de ese modo cambiar el comportamiento [DB03]. Este tipo de arquitecturas han sido hasta cierto punto utilizadas con éxito para implementar sistemas dinámicos. En efecto, facilitan la construcción de sistemas capaces de adaptarse en tiempo de ejecución a los nuevos requisitos impuestos tanto por el usuario como por el desarrollador, obteniendo resultados altamente flexibles. Esto es así por el hecho de que los cambios a ser aplicados en el sistema no se expresan a través de nuevo código, sino a través de *metadatos*, a ser interpretados en ejecución. Estos *metadatos* (ficheros XML o bases de datos específicas) pueden ser modificados fácilmente.

Sin embargo, AOM puede dar lugar a soluciones difíciles de mantener a la hora de incluir nuevas capacidades adaptativas o cambiar el código de las capacidades existentes. Esto ocurre debido a que la interpretación de los *metadatos* y de las acciones asociadas aparece diseminada a lo largo de las clases núcleo. En otras palabras, el código de negocio y el correspondiente a la interfaz se entremezclan con el código que proporciona la adaptación [DYBJ04]. En este sentido, los mecanismos de *Programación Orientada a Aspectos* encapsulan mejor la funcionalidad que entrecruza el código del sistema [CPA04], tal y como se define y defiende en el *Capítulo 6*.

5.4. Resumen y conclusiones del capítulo

La revisión de la literatura en el área de la *sensibilidad al contexto* muestra un extenso trabajo de investigación, así como su aplicación en el desarrollo de gran cantidad de sistemas, entre los que los servicios basados en localización constituyen el área principal de desarrollo. La mayoría de las aplicaciones existentes utilizan muy pocos tipos de contexto, siendo la localización, la identidad y el tiempo los más utilizados. Una posible explicación de esta tendencia es la dificultad para adquirir y procesar información contextual.

Por otro lado, buena parte de los sistemas *sensibles al contexto* son prototipos desarrollados en laboratorios de investigación y entornos académicos. En consecuencia, este tipo de aplicaciones han sido construidas ad hoc, lo que dificulta la construcción de nuevas aplicaciones o la evolución de otras ya existentes. Aún así, su análisis resulta muy valioso no sólo para adquirir una visión general acerca de lo que los investigadores han pretendido alcanzar desde un principio con su desarrollo, sino también para identificar la problemática existente y en base a ello seguir progresando en el campo de la sensibilidad al contexto. En efecto, esa primera fase prospectiva ha generado la necesidad de nuevas infraestructuras.

En un esfuerzo por construir nuevas herramientas estandarizadas para explorar el espacio del problema y desplegar soluciones a gran escala que permitan intercambiar información del contexto surgen las arquitecturas y los middlewares del contexto, estos últimos específicos para aplicaciones distribuidas sobre plataformas heterogéneas. La revisión de estas nuevas herramientas resulta igualmente beneficiosa, con objeto de establecer una base para la investigación futura en el campo.

Las infraestructuras del contexto analizadas han sido seleccionadas con el propósito de seguir una línea coherente con los distintos dominios de aplicación identificados dentro del campo, sin un afán por clarificar la diversidad de enfoques existentes. Para ilustrar la comparativa entre las herramientas seleccionadas se han utilizado los requisitos a cubrir idealmente por una infraestructura de soporte al contexto propuestos por Dey.

Entre los distintos retos identificados en el área de la *computación sensible al contexto*, uno de ellos es el de explorar la manera como utilizar nueva información contextual, así como en reunir y conjugar técnicas de modelado del usuario con el modelado del contexto.

Capítulo 6

Arquitectura software propuesta para el *Motor de Plasticidad Implícita*

"Computer Science is a science of abstraction -creating the right model for a problem and devising the appropriate mechanizable techniques to solve it."

A. Aho - J. Ullman

Este capítulo está destinado a describir la arquitectura propuesta para el *Motor de Plasticidad Implícita*, a ser ubicado en la plataforma cliente como herramienta de tiempo real que da soporte a la *plasticidad implícita*, tal y como se propone bajo la *visión dicotómica de plasticidad* presentada en esta tesis. Antes de pasar a la descripción de la arquitectura se plantea el tipo de problema a cubrir, precisando cuál es la aplicabilidad de aquélla, así como las propiedades a preservar para alcanzar el objetivo planteado, las cuales han sido determinantes en las decisiones de implementación. Se introduce tanto la tecnología adoptada como otros enfoques afines, tratando de justificar la idoneidad de la técnica seleccionada.

Toda esta exposición se hace teniendo presente que la meta propuesta en la presente tesis no acaba en la descripción de una arquitectura software, sino que consiste en el desarrollo de un *framework de aplicación* genérico, instanciable en *Motores de Plasticidad Implícita* para sistemas particulares, al que se le denomina *Framework de Plasticidad Implícita*, y que se presenta en detalle en el *Capítulo 8*.

Una vez descrita la arquitectura, en la sección cuatro se presentan otras iniciativas en el campo del contexto que también han adoptado un enfoque basado en la *Programación Orientada a Aspectos*. Se analizan las similitudes y diferencias.

6.1. Concepción y objetivos del *Motor de Plasticidad Implícita*

El *Motor de Plasticidad Implícita* es la herramienta de tiempo real que da soporte a la *plasticidad implícita* (*Definición 2.7*; *sección 2.2.2*), tal y como se propone bajo la *visión dicotómica de plasticidad* propuesta en esta tesis. Se trata de un motor adaptativo capaz de detectar el contexto y de hacer reaccionar al sistema de manera proactiva ante *variaciones dinámicas en el contexto de uso* (véase *Definición 2.3*; *Capítulo 2 - sección 2.1.3.*), así como también para proporcionar *mecanismos locales de consciencia de grupo* (véase *Definición 2.2*; *Capítulo 2*, *sección 2.1.3*) de acuerdo a la percepción individual de cada miembro del equipo de trabajo, en el caso de entornos colaborativos.

Se trata, por tanto, de la componente responsable de proporcionar *adaptatividad* a la aplicación, tal y como se define este término en el *Capítulo 2* (véase *Capítulo 2*; *sección 2.1.2.*). Un *sistema adaptativo* proporciona adaptaciones dinámicas, esto es, en tiempo real. En este caso están destinadas a aplicar pequeños reajustes o modificaciones específicas tanto en la IU como en ciertos aspectos del comportamiento de la aplicación. Otra de las peculiaridades que caracterizan a los *sistemas adaptativos* es que la adaptación se lleva a cabo de manera automática por parte del sistema, esto es, sin la intervención del usuario. Esta particularidad requiere un proceso de monitorización y/o inferencia para conocer los parámetros que se desean plasmar en la adaptación, previo a su aplicación.

Típicamente, en sistemas de personalización, estos parámetros consisten en los patrones de actuación del usuario y sus preferencias, las cuales pueden ser también cambiantes. No obstante, en esta tesis se aplica un uso generalizado del término, abarcando cualquier aspecto relativo al contexto de uso que pueda variar dinámicamente –recogidos bajo el concepto de *variación dinámica en el contexto de uso*-, y que por tanto el sistema debe ser capaz también de detectar para aplicar la reacción oportuna en el sistema, proporcionando de ese modo la adaptación proactiva perseguida en la plataforma cliente. En particular, los atributos considerados para caracterizar el contexto de uso en esta tesis se presentan en el *Capítulo 2* (véase *sección 2.1.3.*). Como novedad, además de los tres atributos habituales (usuario, entorno y plataforma), entra también en consideración la *consciencia de grupo* [DB92], en los casos en los que se desarrolle una actividad colaborativa. En estos casos, el *Motor de Plasticidad Implícita* se encarga, adicionalmente, de capturar y mantener la *consciencia de grupo particular* (véase definición en la *sección 2.1.3.*), poniendo en funcionamiento algún tipo de *mecanismo local de consciencia de grupo* (véase *Definición 2.2*; *Capítulo 2*, *sección 2.1.3*) con el propósito de fomentar la interacción usuario a usuario entre los miembros del grupo de trabajo, como medida de primer orden para propiciar la colaboración.

De acuerdo a la *Hipótesis 2* –véase *Capítulo 1*; *sección 1.3.*- es preferible resolver este tipo de adaptaciones en el propio equipo donde se lleva a cabo la interacción. Esa es precisamente la misión del *Motor de Plasticidad Implícita*, labor que lleva a cabo de manera autónoma y en segundo plano.

6.2. Premisas iniciales y decisiones de implementación

En esta sección se precisa el tipo de problema a cubrir dentro del campo de la *sensibilidad al contexto*, la aplicabilidad del *Framework de Plasticidad Implícita*, así como las propiedades a preservar teniendo siempre presente la meta de desarrollar un framework de aplicación lo más genérico posible. Se analizan las alternativas más conocidas en el ámbito de las técnicas de *Separación de Conceptos* para llegar a una justificación en cuanto a la adopción de la técnica de *Programación Orientada a Aspectos*. Por último, se introduce una explicación general de esta técnica.

6.2.1. Requisitos de diseño

La mayor dificultad para la construcción de un *Motor de Plasticidad Implícita* en general, y en particular del *Framework de Plasticidad Implícita* es tener que llevar a cabo el tratamiento de múltiples *restricciones de tiempo real* (véase definición proporcionada en el *Capítulo 2*; *sección 2.1.2.*) inherentemente relacionadas entre sí que un *sistema sensible al contexto* (véase *Capítulo 2*; *sección 2.1.3.*) debe capturar, controlar y procesar. El tratamiento de estas restricciones dinámicas generalmente aparece entrelazado con el código núcleo del sistema.

Sin duda, la combinación de cuantas propiedades contextuales nos sea posible favorecerá un entendimiento más preciso de la situación en curso. No obstante, los atributos del contexto manejados suelen ser siempre los mismos, tal y como se analiza en el *Capítulo 5* (véase *Capítulo 5*; *sección 5.3.1.*). Por otro lado, la manera como manejar de forma efectiva toda esta información continúa siendo un problema sin resolver de manera genérica. Como consecuencia, la anticipación a los cambios contextuales no está todavía resuelta de manera satisfactoria, tal y como se expone en el *Capítulo 1* (véase *Capítulo 1*; *sección 1.2.*).

6.2.1.1. Requisitos Funcionales

Con el propósito de desarrollar un sistema capaz de detectar el contexto, a fin de provocar las reacciones apropiadas en el sistema se requiere satisfacer una serie de requisitos previos:

(1) conocer el entorno que rodea al sistema, a fin de caracterizarlo y capturarlo. Este mecanismo implica una habilidad de introspección sobre el comportamiento del usuario y, en su caso, de la actividad colaborativa, así como de percepción del entorno y de los recursos hardware (*sensores*) de acuerdo a los atributos del contexto que se estén considerando en cada caso;

(2) establecer el método para representar e integrar dicho contexto en el funcionamiento del sistema;

(3) establecer los mecanismos necesarios para identificar y detectar las *variaciones en el contexto de uso* y los *cambios contextuales*, establecidos en la fase de diseño, entendiendo estos últimos tal y como se presentan en las *Definiciones 2.3* y *2.4* respectivamente (véase *Capítulo 2*; *sección 2.1.3.*). Por último, es necesario

(4) establecer la operativa necesaria para utilizar efectivamente la información contextual recopilada, con objeto de practicar de manera automática los ajustes y adaptaciones propiamente dichos sobre la IU, y/o sobre el propio funcionamiento del sistema, así como ciertas acciones específicas en favor del desarrollo de la actividad grupal en el caso de entornos colaborativos.

La puesta en práctica de estos mecanismos conlleva tres fases: (1) detección de la información contextual; (2) recogida y (3) uso, como paso definitivo que permite plasmar la adaptación o ejecutar la reacción del sistema en concordancia con el contexto. De manera similar, Calvary et al. en [CCT00] establecieron un proceso en tres pasos: (1) *reconocimiento de la situación*, que incluye tanto la detección del contexto de uso a través de sensores como la consciencia de la magnitud e implicación del cambio contextual; (2) *computación de una reacción*, que comporta la identificación de las posibles reacciones y la selección de la candidata de acuerdo a un *coste de migración* aceptable (véase *sección 2.1.5.*); y (3) *ejecución de la reacción*, tal y como se presenta en el *Capítulo 3* (véase *sección 3.3.4.1*).

6.2.1.2. Requisitos No Funcionales

A continuación se precisan los criterios de aplicabilidad que han estado presentes en la construcción del *Framework de Plasticidad Implícita*, así como las propiedades y metas a satisfacer en su desarrollo, de cara a conseguir que sea lo más genérico posible.

6.2.1.2.1. Premisas iniciales

Antes de entrar en detalle en la explicación de la estructura software propuesta y llegar a comprender qué requisitos no funcionales nos interesa hacer prevalecer, es importante dejar claras una serie de premisas.

Tal y como ha sido mencionado, el artefacto software final desarrollado en esta tesis es un framework genérico fácilmente personalizable (instanciable) en *Motores de Plasticidad Implícita* específicos para sistemas particulares con unas necesidades contextuales concretas. Este proceso de instanciación consta de dos pasos:

(1) especialización de los componentes del framework para el sistema destino

(2) acoplamiento de los componentes personalizados con la aplicación utilizando determinados puntos de "enganche" (del término inglés *hook*).

El framework resultante se denomina *Framework de Plasticidad Implícita*, el cual se presenta en detalle en el *Capítulo 8*.

Cabe puntualizar aquí que los componentes que forman parte del framework están dedicadas exclusivamente a la funcionalidad adicional relativa al contexto que incluye los pasos de detección de la información contextual, recogida y uso de la misma, descritos en la sección anterior. En ningún momento se plantea como un framework del que derivar también la funcionalidad núcleo del sistema. Eso limitaría considerablemente el dominio de aplicación y, sobretodo, trazaría una estructura rígida sobre la que construir este tipo de aplicaciones, restringiendo por tanto en gran medida el ámbito de aplicación del framework.

Bajo este punto de vista, en el que se aborda la construcción de *aplicaciones sensibles al contexto* desde un principio, la funcionalidad auxiliar relativa al contexto constituye un requisito temprano, esto es, presente desde las primeras fases de desarrollo del sistema. Por el hecho de tratarse de un requisito que, como ya se ha avanzado anteriormente y en el que se irá insistiendo, dado que constituye una de las hipótesis de esta tesis (véase *Capítulo 1*; *sección 1.3.*), aparece entrelazado con el código núcleo del sistema, debería ser considerado y abordado como un *aspecto temprano* (del término inglés "early aspect"). Se denomina *aspecto temprano* a una propiedad que ya en las fases tempranas del desarrollo de software (fases de análisis de requisitos y definición de la arquitectura) se identifica como un requisito inherentemente relacionado con otros, y que por tanto inevitablemente tiene un fuerte impacto sobre ellos en gran parte de los componentes del sistema. Se suele utilizar la expresión de que *"se entrecruza con otros requisitos y módulos del sistema"*. Por ese motivo este tipo de requisitos se definen como *conceptos transversales* (término proveniente del inglés *crosscutting concern*; véase *Definición 6.1* más adelante).

Cuando estos requisitos ya aparecen en las fases tempranas de desarrollo, no es suficiente con atacar la problemática derivada (analizada más adelante; véase *sección 6.2.3.1.*)

en la fase de implementación, recurriendo a los mecanismos de la *Programación Orientada a Aspectos* –técnica adoptada para la implementación del framework-, sino que es más conveniente abordarla ya desde la fase de análisis, aplicando técnicas de Desarrollo de Software Orientado a Aspectos (del término inglés *Aspect-Oriented Software Development*, abreviadamente AOSD). AOSD[1] constituye una disciplina y área de investigación incipiente dentro en la ingeniería del software. Promueve el uso de *aspectos* (definidos más adelante –véase *sección 6.2.3.3.1.*-) en todas las fases del desarrollo del software, considerando el impacto de estos requisitos ya desde las fases tempranas (de ahí el nombre de *aspectos tempranos*). Esto es lo que distingue AOSD de la *Programación Orientada a Aspectos* (POA en lo sucesivo), en la que tan sólo se consideran este tipo de requisitos a nivel de programación. Se trata de un área mucho más consolidada, que fue la que dio origen al tratamiento de los *conceptos transversales*.

Ese no es el enfoque adoptado en esta tesis para el *Framework de Plasticidad Implícita*, puesto que no aborda la construcción de *aplicaciones sensibles al contexto* desde un principio. Los requisitos problemáticos (los *conceptos transversales*) no aparecen hasta que se plantea incrementar el sistema con la funcionalidad del contexto, una vez completada la implementación de la aplicación base. Bajo este enfoque de aplicabilidad tan sólo es factible aplicar técnicas de separación de *conceptos transversales* a nivel de programación, puesto que las aplicaciones a tratar ya se encuentran completamente implementadas. Se trata, por tanto, de disponer de los módulos o componentes software necesarios para incorporar la funcionalidad de *sensibilidad al contexto* en aplicaciones ya desarrolladas. Estas componentes deben ser fácilmente personalizables a distintos sistemas, y también de fácil acoplamiento con la aplicación.

Este enfoque amplía el campo de aplicación respecto al anterior, siempre y cuando se preserve la independencia con el sistema y con el dominio. En contrapartida tiene un precio a pagar: es necesario llevar a cabo un paso de integración entre la aplicación ya implementada y los componentes del framework desarrollados de manera independiente. Se trata del paso identificado anteriormente como de acoplamiento, el segundo a llevar a cabo en la instanciación del framework. No se trata de un paso sencillo, puesto que se requiere un conocimiento profundo de la aplicación destino, o como mínimo de aquellos aspectos que la nueva funcionalidad interfiere o modifica. Además, debe aplicarse un enfoque que permita desacoplar al máximo ambas componentes (aplicación y framework instanciado) para lograr genericidad e independencia del sistema.

De ahí que los puntales que han gobernado la construcción del framework genérico han sido (1) el de máxima genericidad, con objeto de obtener la máxima reutilización, a

[1] Aspect-Oriented Software Development Community http://aosd.net/

fin de que los componentes proporcionados por el framework sean lo más independientes posible del dominio y del sistema; y (2) el mínimo impacto sobre la aplicación a la hora de llevar a cabo el acoplamiento. Se persigue mantener intactos en la medida de lo posible tanto la estructura como el código de la aplicación destino. Esto significa que los puntos de "enganche" con los componentes adicionales debe ser lo más superficial posible.

En efecto, si se consigue preservar la estructura inicial del sistema (propiedad de *transparencia*), su readaptación a un conjunto de necesidades contextuales distintas no resulta traumático, limitándose tan sólo a repetir de nuevo el proceso de instanciación de las nuevas componentes del contexto.

En resumen, el framework que se propone en esta tesis está concebido para poder dotar a un sistema que no es *sensible al contexto* de esta habilidad extra, sin interceptar en su estructura original ni en su código, consiguiendo aumentar el sistema de forma transparente. Sólo de ese modo se consigue una máxima *reutilización*, no sólo en los componentes relativos al contexto, sino también en el sistema subyacente, el cual de este modo puede irse adaptando a distintas necesidades contextuales, acoplando y desacoplando los componentes adecuados, oportunamente personalizados en cada caso, según las necesidades que se vayan presentando. Esto refuerza el concepto de plasticidad, de acuerdo a la definición presentada en el *Capítulo 2* (véase *Capítulo 2*; *sección 2.1.1.*):

> *"La capacidad de un sistema interactivo de soportar variaciones en el contexto de uso mientras se preserva la usabilidad" [CCT00].*

Esta idea evoca el concepto expresado mediante el término inglés *pluggability*. Puede entenderse como la facilidad para generar distintas versiones de una misma aplicación, dependiendo en este caso de las necesidades contextuales cambiantes.

Por último, remarcar que el objetivo de un framework, por definición, es el de facilitar la labor al desarrollador de aplicaciones, en este caso de *aplicaciones sensibles al contexto*, con el propósito de liberar al programador de tener que pensar en toda la funcionalidad en su conjunto.

Para ello se debe procurar que la instanciación de los componentes del framework en el sistema en cuestión consista en un paso lo más natural e independiente posible. Para conseguirlo deben cumplirse dos premisas: (1) que la mayoría de la funcionalidad *sensible al contexto* sea parte integrante del framework, es decir, esté ya diseñada e implementada; y (2) que el acoplamiento no suponga ningún tipo de impacto sobre la aplicación. Este

último paso debe consistir tan sólo en resolver los puntos de "enganche" entre el framework, una vez personalizado, y la aplicación.

Éstas son las premisas de partida para el desarrollo del *Framework de Plasticidad Implícita*. En el *Capítulo 8* se presentan en detalle las técnicas aplicadas para lograr un nivel aceptable de genericidad, reutilización y transparencia.

6.2.1.2.2. Metas y propiedades a satisfacer

Con objeto de satisfacer las premisas de partida presentadas anteriormente, las cuales condicionan el enfoque a seguir, existen una serie de propiedades o principios software que deben ser garantizados en el diseño del *Framework de Plasticidad Implícita*, tal y como ha sido planteado, a fin de alcanzar la versatilidad necesaria para la personalización del mismo en distintos sistemas y necesidades contextuales.

A pesar de que estas propiedades ya han ido apareciendo implícitamente a lo largo de la sección anterior, se enumeran a continuación de forma detallada. Todas ellas están íntimamente relacionadas entre sí.

(1) *Transparencia* en el sentido de que debe procurarse que la estructura y el código del sistema subyacente no se vean afectados por la integración de los mecanismos de adaptación al contexto, los cuales deberán ser estructurados en una o varias capas adicionales. En efecto, para conseguir una reutilización en ambos sentidos, esto es, que por un lado el sistema sea fácilmente amoldable a distintas necesidades contextuales, y por otro lado que los componentes que otorgan esa habilidad extra al sistema puedan ser fácilmente personalizadas a tantos sistemas como sea posible, debe procurarse reducir al máximo las dependencias entre capas. Como ya se ha mencionado, se trata de ofrecer a través del framework componentes o unidades de programa lo más independientes y genéricas posible, con objeto de proporcionar la máxima reutilización.

(2) *Ortogonalidad*. Muy relacionada con la propiedad de transparencia, en el sentido de independizar el tratamiento de las distintas *restricciones de tiempo real* no sólo con respecto al sistema subyacente, sino también entre sí. Esta propiedad es esencial para que las capacidades adaptativas puedan ser manejadas independientemente entre sí, facilitando la generación de distintas versiones del sistema. Sólo de ese modo pueden fácilmente agregarse o descartarse en las distintas versiones del sistema, conforme las necesidades contextuales vayan evolucionando, alcanzando esa *'pluggability'* a la que se ha hecho referencia, y siguiendo las directrices de las tecnologías de *Separación de Conceptos*.

El principio de *Separación de Conceptos* fomenta la máxima modularidad en los requisitos extra-funcionales, y ha sido reconocida como un mecanismo fundamental para

manejar la complejidad de los sistemas software [Bir04]. Esta filosofía repercute en favor de la *reutilización*.

De hecho, la ortogonalidad es especialmente importante en aquellos sistemas en los que se manejan y procesan un elevado número de requisitos (a los que suele denotarse con el término inglés *concern*). Este es el caso, precisamente, de las aplicaciones móviles en general [EVP00], y de las *aplicaciones sensibles al contexto* en particular [ZGJ04]. De hecho, el manejo de gran cantidad de requisitos es especialmente crítico cuando éstos irremediablemente (1) se entrelazan (enmarañan) entre sí debido a su estrecha e inherente relación; y (2) se dispersan a lo largo de la funcionalidad núcleo. A aquellos requisitos que a pesar de seguir un enfoque basado en la *Programación Orientada a Objetos* (POO en adelante) manifiestan esta problemática, son denominados comúnmente *conceptos transversales* (véase más adelante *Definición 6.1*).

(3) *Reutilización* de los mecanismos de adaptación en diversos sistemas con necesidades contextuales cambiantes. El cumplimiento de las dos propiedades anteriores hace que los límites para instanciar el framework a distintos sistemas y circunstancias contextuales tan sólo vengan impuestos por la oferta de componentes ofrecida por el propio framework, esto es, por la variedad de módulos o componentes de que consta. Igualmente, cuanto más independientes sean entre sí los módulos proporcionados por el framework, mayor número de combinaciones de necesidades contextuales en sistemas concretos podrán ser configuradas.

Con el propósito de garantizar estas tres propiedades se deduce que lo más conveniente es aplicar alguna de las tecnologías pertenecientes a la categoría de *Separación de Conceptos*, las cuales centran sus esfuerzos en descomponer los programas en función de los conceptos que en él son manejados, a fin de evitar tanto como sea posible el solapamiento de funcionalidad entre los distintos módulos obtenidos, así como de minimizar las dependencias entre ellos. En este contexto, se entiende por '*concepto*' (que se corresponde con el término inglés *'concern'*), cualquier requisito, pieza de interés o foco de atención de un sistema. La meta de una tecnología de *Separación de Conceptos* es la de encapsular los requisitos extra-funcionales en módulos separados entre sí y con respecto a la funcionalidad núcleo.

En el caso que nos ocupa, los requisitos extra-funcionales a modularizar son precisamente los encargados del tratamiento de las *restricciones de tiempo real* y la consecuente adaptación de la aplicación, así como los *mecanismos locales de consciencia de grupo* en el caso de entornos colaborativos.

En las sub-secciones siguientes se presentan algunos de los enfoques de *Separación de Conceptos* que han ido surgiendo, se analizan sus similitudes y diferencias, entre ellos y

con respecto a la técnica adoptada, la POA, con la intención de justificar el por qué de su adopción para la implementación del *Framework de Plasticidad Implícita*.

6.2.2. Enfoques para la Separación de Conceptos

El tópico de *Separación de Conceptos* ha constituido un área de investigación muy activa en los últimos años. Los diversos enfoques iniciales fueron tomando distintas terminologías en cada uno de los grupos de desarrollo, hasta que finalmente fueron agrupadas sobre este mismo paraguas denominado *Separación de Conceptos*, y el cual está inspirado en el principio de *"divide y vencerás"*. Los distintos enfoques coinciden en perseguir la reducción de la complejidad de los sistemas software a través de la abstracción. La abstracción comporta separación, lo que repercute en una programación sustancialmente menos compleja y más efectiva.

Adicionalmente, la *Separación de Conceptos* introduce, como consecuencia de una adecuada separación, una disminución del acoplamiento entre los distintos requisitos del sistema, otro de los principios fundamentales de un buen diseño software. Eso repercute en una independencia sustancial entre módulos, de manera que los cambios en un requisito afectan mínimamente en los otros.

Los investigadores en el campo han estudiado varias técnicas y enfoques para conseguir modularizar la implementación de los distintos requisitos del sistema bajo el tópico general de *Separación de Conceptos* (o también *separación de competencias*), tratando de subsanar las limitaciones de los paradigmas tradicionales, en particular de la POO. Estos desarrollos extienden el modelo puro de objetos agregando nuevos conceptos y mecanismos necesarios para su composición. Algunos persiguen los mismos objetivos y otros no.

Las más importantes hoy en día, además de la POA son la *programación meta-nivel*, la *programación orientada a sujetos* y los *filtros de composición* (del término inglés *composition filters*). A continuación se presenta una breve explicación de cada una de ellas, seguida de una discusión final.

6.2.2.1. Filtros de composición

Los filtros de composición (FC) fueron introducidos por Aksit et al. en [ABV92] de la Universidad de Twente, Holanda. Los FC se consideran pertenecientes a la primera generación de lenguajes *orientados a aspectos* [HL95].

6.2.2.1.1. Motivación

Surgen con el propósito de subsanar las dificultades que aparecen al expresar la coordinación de los mensajes en el modelo de objetos tradicional. Se puede decir que uno de los problemas del modelo de objetos convencional es la ausencia de mecanismos adecuados para separar la funcionalidad del código para la coordinación de los mensajes. FC extiende el modelo convencional de objetos agregando los denominados *filtros de mensajes*. Por ejemplo, para expresar la sincronización de un objeto se requiere algún mecanismo para inyectar el código referente a la sincronización en todos los métodos que deben ser sincronizados. Extender una clase con nuevos métodos mediante herencia podría requerir cambios en el esquema de sincronización. Este y otros problemas relacionados son conocidos como *anomalías de herencia* [MY93].

6.2.2.1.2. Descripción

Este modelo proporciona control sobre los mensajes recibidos y enviados por un objeto. En particular, un filtro especifica condiciones para la aceptación o rechazo de mensajes, y determina la acción a llevar a cabo.

Bajo este enfoque, un objeto es definido como una instancia de una clase que se compone de, además de los miembros habituales (métodos, atributos, etc.), uno o más filtros. Los filtros son también objetos, y por lo tanto constituyen instancias de clases filtro. Cada clase filtro es responsable de manejar todos los aspectos relacionados con su requisito asociado.

En general, cualquier requisito que requiera interceptar mensajes o extender métodos con acciones previas y/o posteriores a la ejecución del método, puede ser expresado usando FC.

La mayoría del proceso de recomposición se lleva a cabo en tiempo de ejecución. Los puntos de unión son los mensajes dinámicos que llegan y parten de un objeto.

El lenguaje principal es un lenguaje de POO, y el mecanismo de los filtros *de composición* proporciona un lenguaje utilizado para controlar ciertos aspectos relacionados con la sincronización y comunicación.

6.2.2.1.3. Modo particular de alcanzar separación de conceptos

La separación de conceptos se alcanza definiendo una clase filtro por cada requisito. En efecto, este mecanismo proporciona la posibilidad de atrapar tanto la recepción como el envío de mensajes, y de ejecutar ciertas acciones previas a la ejecución del método. En

consecuencia, el código resultante está separado entre el propósito especial (el encapsulado en el filtro) y el requisito base (el del método).

6.2.2.2. Programación meta-nivel

La programación meta-nivel es un paradigma bien conocido en el campo de la ingeniería del software que ha sido documentado en numerosas publicaciones. Las más representativas son las siguientes: [Mae87], [Smi84], [Zim96].

6.2.2.2.1. Motivación

El campo en el que esta técnica está teniendo más aceptación es en el de desarrollo de middlewares, ofreciendo una potente alternativa a los diseños monolíticos, que producen implementaciones inflexibles y difíciles de adaptar en dominios de aplicación donde los recursos son limitados y los requisitos cambiantes. Esta técnica ofrece la habilidad de reconfiguración, así como también la posibilidad de mejorar la interoperabilidad de los middlewares y, por supuesto, una mejor separación de conceptos.

6.2.2.2.2. Descripción

La meta-programación es la capacidad que tienen los programas de razonar acerca de sí mismos. Un lenguaje permite la meta-programación cuando tiene la capacidad de razonar sobre programas escritos en ese mismo lenguaje.

Una de las propiedades fundamentales de la *programación meta-nivel* es que el programador tiene acceso a las estructuras que representa un programa, de manera que se dispone de una representación del programa en tiempo de ejecución. A esta representación se le denomina *auto-representación*, la cual es proporcionada por el nivel meta y está *causalmente conectada* al comportamiento subyacente. Estos términos se describen en detalle en el *Capítulo 4* (véase *Capítulo 4*; sección 4.1.4.1.4.).

Una arquitectura meta-nivel proporciona un nivel base y uno o más meta-niveles que proporcionan control sobre la semántica del lenguaje base y la implementación. La interfaz entre el programa base y los niveles superiores se realiza a través de los denominados *protocolos meta-objeto*. Son interfaces que proporcionan al programador la habilidad de modificar incrementalmente el comportamiento del lenguaje y su implementación [KdRB91]. El comportamiento de los objetos del nivel base, responsables de la funcionalidad núcleo, puede por tanto ser aumentado y adaptado por parte de los meta-objetos, de acuerdo a determinados requisitos. Por ejemplo, el envío de mensajes puede ser atrapado por un

meta-objeto, el cual puede comprobar ciertas restricciones de sincronización previo a su entrega efectiva [HL95].

Las construcciones básicas del lenguaje de programación, tales como clases o invocación de objetos se describen en el meta-nivel y pueden ser extendidas o redefinidas a través de meta-programación. Cada objeto está asociado con un meta-objeto, el cual es responsable de la semántica de las operaciones en el objeto base. Los *protocolos meta-objeto* proporcionan vistas de la computación que ningún componente del nivel base puede percibir [Rei00].

El lenguaje más popular que implementa los conceptos de la programación meta-nivel es CLOS (Common Lisp Object System) [Kos90].

6.2.2.2.3. Modo particular de alcanzar separación de conceptos

El modo como se alcanza la separación de conceptos es atrapando el envío y la recepción de mensajes, los meta-objetos tienen la oportunidad de ejecutar requisitos extra, los cuales se implementan en los meta-objetos. Esto permite que el código del nivel base pueda centrarse en los requisitos estrictamente funcionales.

6.2.2.3. Subject-Oriented Programming

Subject-Oriented Programming (SOP) es una tecnología de composición de programas de IBM que fue propuesta por Harrison y Ossher en [HO93]. Es muy similar a la POA, aunque no representa exactamente el mismo concepto [LAC05].

6.2.2.3.1. Motivación

SOP constituye una extensión al paradigma de POO con el fin de tratar diferentes perspectivas subjetivas de los objetos modelados. Por ejemplo, para un mismo objeto árbol distintos sujetos tienen diferentes perspectivas del mismo objeto. Así, modelado desde el punto de vista de un leñador tendrá atributos como 'precio de venta' o 'tiempo de talado', mientras que para un ecologista un posible atributo de interés puede ser el de 'valor estimado', desde el punto de vista ecológico. Al igual que un mismo objeto modelado desde diferentes puntos de vista posee diferentes atributos, lo mismo ocurre con su comportamiento. A cada una de las *perspectivas subjetivas* acerca de un mismo objeto se le llama *subject*.

Sin embargo, los diferentes contextos de utilización no son la única razón para usar *subjects*. Este enfoque también sirve para tratar los conflictos que pudieran surgir en la integración cuando el software es desarrollado con un alto grado de independencia.

6.2.2.3.2. Descripción

Un *subject* es una colección de clases y/o fragmentos de clases relacionados por herencia u otras relaciones definidas por el *subject*. Por lo tanto, el *subject* es un modelo de objetos completo o parcial. Se expresan utilizando un lenguaje orientado a objetos estándar.

Los *subjects* pueden ser simples o compuestos, siguiendo en éste último caso las siguientes reglas de composición: (1) reglas de correspondencia (especifican, si procede, la correspondencia entre clases, atributos y métodos de diferentes *subjects*) y (2) reglas de combinación (establecen cómo dos *subjects* deben ser combinados). El resultado de la composición es un nuevo *subject*que combina la funcionalidad de los *subjects* componentes.

SOP es un método de composición de programas que soporta la construcción de sistemas orientados a objetos como composiciones de *subjects*, extendiendo de ese modo los sistemas originales o integrando distintos sistemas entre sí.

La composición de *subjects* combina jerarquías de clases para producir nuevos *subjects* que incorporan funcionalidad de los *subjects* existentes.

6.2.2.3.3. Modo particular de alcanzar separación de conceptos

SOP divide un sistema en *subjects*, los cuales se componen a través de reglas de composición que introducen flexibilidad a los programas *orientados a objetos*, proporcionando nuevas oportunidades para desarrollar y modular programas a gran escala.

6.2.2.4. Discusión

Los *filtros de composición* y la *programación meta-nivel* tienen en común el hecho de que proporcionan un mecanismo genérico para interceptar mensajes. Los *protocolos meta-objeto* llevan a cabo la interceptación en el meta-nivel. Los *filtros de composición* atrapan los mensajes a través del mecanismo de filtros embebido. En ambos casos la interceptación se realiza en tiempo de ejecución, lo que supone una reducción de la eficiencia en comparación con las técnicas convencionales. Cabe añadir que el uso de *protocolos meta-objeto* complica mucho la programación.

Una ventaja de los *filtros de composición* con respecto a la *programación meta-nivel* es que son ortogonales entre sí y que pueden ser combinados libremente. La *programación meta-nivel* no resuelve de forma elegante la separación entre los distintos requisitos en el meta-nivel. En contrapartida, la *programación meta-nivel* tiene la ventaja de que la separación de conceptos no viene impuesta por el modelo. En otras palabras, programar los requisitos auxiliares en el meta-nivel constituye una estrategia a la que los programadores

pueden optar o no sin más. En cambio, los *filtros de composición* requieren construcciones específicas del lenguaje para alcanzar la separación de conceptos y, en consecuencia, deben ser introducidas para cada nuevo requisito auxiliar.

Ambas técnicas han demostrado soportar la separación de conceptos no sólo a nivel conceptual, sino también a nivel de implementación. No obstante, no queda resuelta la composición de todo ese código y bloques de datos en un producto final coherente, dando lugar al problema de composición de componentes software, tal y como se identifica y discute en [HL95]. Sorprendentemente, cuando coexisten múltiples requisitos el problema de composición, aunque en un nivel de abstracción superior, se vuelve recurrente. La POA, sin embargo, mantiene su claridad a pesar de la coexistencia de diversos requisitos.

Por otro lado, los meta-lenguajes son de más bajo nivel que los lenguajes de *aspectos* en los que los *puntos de enlace* son equivalentes a los "ganchos" de la *programación meta-nivel*. No obstante, estos sistemas se pueden utilizar como herramienta para la POA.

Por lo que respecta a SOP, la diferencia clave con respecto a POA es que mientras que los *aspectos* de la POA tienden a ser propiedades que afectan al rendimiento o la semántica de los componentes, los *subjects* de la SOP son características adicionales que se agregan a otros *subjects* [LAC05].

Por otro lado, SOP constituye una técnica de separación de conceptos simétrica. Los paradigmas simétricos sugieren componer las distintas dimensiones del sistema. A la hora de aplicar esta técnica en el campo de la *sensibilidad al contexto* surgen ciertos interrogantes difíciles de resolver: ¿La *sensibilidad al contexto* constituye una dimensión del sistema?, ¿Se trata de una o muchas dimensiones?. Existen *restricciones de tiempo real* muy acotadas que no parecen tener suficiente envergadura como para constituir una dimensión. Por otro lado, si reunimos todos esos pequeños requisitos en uno solo el resultado sería que todos ellos aparecerían acoplados de manera tal que no podrían ser tratados ortogonalmente. En POA la identificación de los distintos requisitos que componen el programa proporciona mayor libertad.

Por último cabe destacar los requisitos que Czarnecki establece para los mecanismos de composición en [CE00], los cuales tan sólo se cumplen en el paradigma de POA. Estos requisitos ayudan a analizar y comprender su evolución en términos de composición. Se pueden sintetizar de la forma siguiente:

(1) acoplamiento mínimo;

(2) diferentes tiempos y modos para el enlace (*binding*) de *aspectos*; y

(3) adición no invasiva de *aspectos* respecto del código existente.

POA proporciona muchos nuevos conceptos interesantes, algunos de ellos pertenecientes a la *programación reflexiva* y *arquitecturas meta-nivel*. No obstante, la POA es considerablemente más fácil de entender que la *programación meta-nivel* y está teniendo un mayor número de seguidores [KHH+01b].

6.2.3. Técnica adoptada: Programación Orientada a Aspectos

La POA es una metodología de *separación de conceptos* que proporciona mecanismos que hacen posible abstraer los distintos *conceptos transversales* del sistema para posteriormente componer un todo, tratando de implementar un sistema software de forma eficiente y fácil de entender.

POA es un desarrollo que sigue al paradigma de la orientación a objetos, y como tal, soporta la descomposición *orientada a objetos*, además de la procedural y la funcional. Pero, a pesar de esto, POA no se puede considerar como una extensión de la POO, ya que puede utilizarse con otros estilos de programación [Rei00]. El estado actual de la investigación en POA es análogo al que había hace veinte años con respecto a la POO [Rei00].

6.2.3.1. Motivaciones de su aparición

Hoy en día es habitual que en el desarrollo de un proyecto software, además de tener que abordar la funcionalidad núcleo (*código de negocio*) para la que es concebido, sea necesario abordar también un conjunto de requisitos auxiliares de tipo no funcional (requisitos extra-funcionales, o también *código de soporte*). Estos últimos constituyen características globales que se exigen adicionalmente a los productos software, y que aparecen comúnmente en la mayoría de proyectos. Ejemplos de este tipo de requisitos son: integridad, autenticación, seguridad, *logging* o seguimiento de mensajes, persistencia, concurrencia, gestión de memoria, comprobación y control de errores, realización de sistemas de trazas, monitorización, sincronización, etc. Se suele describir esta circunstancia haciendo uso de la expresión *"espacio de requisitos n-dimensional"*.

Por el hecho de tratarse de características globales, influyen y reciben la influencia de diversas componentes del sistema, en ocasiones sin relación aparente entre sí, las cuales escapan a la ordenación tradicional por capas. Por consiguiente, su tratamiento suele afectar a gran parte de la funcionalidad núcleo, lo que provoca que el código correspondiente aparezca disperso (también referenciado como código transversal o entrecruzado) a lo largo de los módulos del sistema, siendo a veces incluso redundante, además de enmarañarse con el código núcleo. Tanto es así que a los requisitos extra-funcionales cuyo código, a pesar

de utilizar la POO presenta estos síntomas, se les denomina *conceptos transversales* (del término inglés *crosscutting-concerns*) [KLM+97]. Los síntomas a los que se está haciendo referencia se identifican con estos términos:

- *Dispersión de código* (del término inglés 'code scattering') en el tratamiento de un requisito auxiliar. El código encargado del tratamiento de un requisito, más allá de encontrarse localizado en un módulo, se extiende a lo largo del código núcleo, implicando un volumen importante de componentes y generando un alto acoplamiento.
- *Enmarañamiento de código* (del término inglés 'code tangling'). El núcleo del sistema, además de encargarse de la funcionalidad principal, debe interactuar y llevar a cabo el tratamiento de una serie de requisitos adicionales, provocando 'contaminación de código' y falta de *ortogonalidad*. Esta circunstancia se traduce en una pérdida de cohesión importante en los componentes de software.

Llegado a este punto es conveniente introducir la siguiente definición:

Definición 6.1 (concepto transversal): requisito extra-funcional que mediante la aplicación de la Programación Orientada a Objetos no puede ser abstraído elegantemente en una unidad funcional y, como consecuencia, aparece inevitablemente disperso a lo largo de la mayoría de los módulos del sistema, enmarañando la funcionalidad núcleo y provocando los consecuentes efectos negativos en la reutilización, legibilidad y flexibilidad del código. □

Se trata de requisitos que resulta imposible o muy difícil de abstraer en una clase concreta, de manera que se reparten en forma de código repetido en diversos módulos, a pesar de ser conceptualmente independientes entre sí. Estos módulos, por tanto, pierden cohesión y acaban teniendo un fuerte acoplamiento. En definitiva, este problema se produce por el hecho de trabajar en un espacio de implementación uni-dimensional, a pesar de afrontarse a un *"espacio de requisitos n-dimensional"*. Suele utilizarse la siguiente expresión para hacer referencia a esta problemática:

> *"el espacio de requisitos es n-dimensional, mientras que el espacio de implementación es uni-dimensional" [Lad02].*

refiriéndose como espacio de implementación uni-dimensional a la POO. Esta dimensión única es la que se destina a la funcionalidad núcleo, de manera que suele aparecer forzosamente acompañada de la implementación del resto de los requisitos extra-funcionales, siendo aquélla la que domina a lo largo de la implementación. En la jerga del campo se

utiliza la expresión "*tiranía de la dimensión dominante*" que fue acuñada por Tarr et al. en [TOHS99] para hacer referencia a esta circunstancia. El resto de requisitos, por tanto, no son resueltos ortogonalmente. El resultado es un mapeo requisitos-implementación muy poco adecuado.

Todo este razonamiento nos lleva a la siguiente conclusión:

> *La POO no resuelve satisfactoriamente la multiplicidad de requisitos extra-funcionales, presentando los síntomas descritos de dispersión y enmarañamiento de código, situación que tiene claramente un impacto negativo en la calidad del software que se manifiesta en una trazabilidad pobre, una baja productividad y reutilización, así como la imposibilidad de hacer evolucionar el código* [Rei00].

Otra de las consecuencias con las que se enfrenta la POO es la que se recoge bajo el concepto del *Dilema del Arquitecto* [Lad03], que se presenta a continuación.

Una buena arquitectura software considera tanto los requisitos presentes como futuros, con objeto de evitar los conocidos 'parches de código'. No obstante, esto plantea un problema, pues es complejo conocer a priori todos los requisitos. Obviar futuros requisitos provoca tener que modificar muchas partes del sistema en lo sucesivo. Por otro lado, centrarse demasiado en requisitos de baja probabilidad puede conducir a un código confuso. Esto plantea un dilema, conocido como el *dilema del arquitecto*, el cual puede ser enunciado de este modo:

Dilema del Arquitecto (Laddad, 03) *Dificultad para conocer a priori todos los requisitos que el sistema tendrá que abordar en el futuro, puesto que no acostumbran a estar claros desde un principio. Esta circunstancia fuerza en muchos casos a que el desarrollador tome decisiones de diseño en las fases tempranas del desarrollo, en muchos casos precipitadas, dando lugar a dos posibles repercusiones: un sobre-diseño o saturación del sistema, o bien un sub-diseño, como resultado de obviar ciertos requisitos.*

En resumen, los problemas que plantea la POO suscitan la necesidad de proponer y utilizar nuevas metodologías que permitan un mapeo más natural entre los requisitos del sistema y las construcciones de programación, repercutiendo en una mayor facilidad para crear sistemas con una complejidad creciente. La POA intenta llenar este vacío.

6.2.3.2. Un poco de historia

A principios de los años 90 investigadores del laboratorio de Xerox PARC (Palo Alto Research Center), liderado por Gregor Kiczales dedicaban sus esfuerzos a examinar y de-

tectar las limitaciones de la POO, como consecuencia de la creciente complejidad de los sistemas software. En esos momentos se creó un equipo de trabajo financiado por el DARPA (Defense Advanced Research Projects Agency). Sin embargo, previamente el grupo *Demeter*[2] ya había estado utilizando ideas *orientadas a aspectos*. El trabajo del grupo *Demeter* estaba centrado en la *Programación Adaptativa* (PA), introducida alrededor de 1991, la cual puede considerarse como una instancia temprana de la POA. Esta programación divide los programas en varios *bloques de corte*, constituyendo un gran avance dentro de la tecnología de software, que se basa en el uso de autómatas finitos.

La relación entre la POA y la PA surge de la Ley de Demeter: *"Sólo conversa con tus amigos inmediatos"*. Esta ley, inventada en 1987 en la Northeastern University, y popularizada en libros de Booch, Budd, Coleman, Larman, Page-Jones, Rumbaugh, entre otros, es una simple regla de estilo en el diseño de sistemas orientados a objetos que viene a resaltar la importancia de obtener software lo más modular posible [LAC05].

La primera definición del concepto de *aspecto* fue publicada en 1995, también por el grupo Demeter, y se describía de la siguiente forma: *"unidad que se define en términos de información parcial de otras unidades"*.

Cristina Lopes y Karl Lieberherr, del grupo Demeter, empezaron a trabajar con Gregor Kiczales y su grupo. A Lopes y Kiczales no les gustó el nombre *Programación Adaptativa* e introdujeron el término *Programación Orientada a Aspectos* con su propia definición y terminología en 1996 [Kic96]. La literatura considera a Gregor Kiczales como el creador de este nuevo paradigma [Rei00].

Al crecer la POA como paradigma fue posible definir la PA como un caso especial de la POA [HL95], y por lo tanto se considera que *"la PA es igual a la POA con grafos y estrategias transversales"*.

En 1997 se celebra el primer evento en la disciplina: el ECOOP (European Conference on Object-Oriented Programming). Desde entonces ha tenido una gran presencia en entornos académicos, y actualmente comienza a estar cada vez más presente en entornos de desarrollo reales. Uno de los avances fue la creación del lenguaje *AspectJ* [KHH+01a] en 1998, una extensión al lenguaje convencional Java, como primera implementación de un lenguaje de POA que ofrece nueva sintaxis para definir, crear y utilizar *aspectos*. En 2002 la empresa desarrolladora Xerox PARC transfirió *AspectJ* a la comunidad de código abierto *Eclipse Foundation*[3], que sigue soportando el proyecto en la actualidad, convirtiéndose en el lenguaje de POA más popular y consolidado. Actualmente *AspectJ* puede ser utilizada en casi cualquier herramienta y entorno de programación.

[2] http://www.ccs.neu.edu/research/demeter/biblio/dem-book.html
[3] http://www.eclipse.org/

6.2.3.3. Puntos clave de la Programación Orientada a Aspectos

Entre los objetivos que se propone la POA están principalmente el de separar con claridad el tratamiento de los *conceptos transversales* en módulos bien localizados y el de minimizar las dependencias inherentes entre ellos, contribuyendo a una pérdida importante del acoplamiento entre los distintos requisitos de un sistema software. Esto se consigue añadiendo una nueva dimensión en la que extraer ortogonalmente ese tipo de requisitos.

6.2.3.3.1. Fundamentos de la POA

La POA, al igual que las otras técnicas de *separación de conceptos*, ataca la problemática de la POO anteriormente presentada, elevando su nivel de abstracción con el fin de alcanzar un mapeo natural entre los requisitos del sistema y los componentes, módulos o unidades de programa en que se divide el sistema.

Para ello introduce una nueva abstracción: la entidad *aspecto*. Consiste en un concepto similar al de una *clase* en POO, pero a un nivel superior de abstracción. Cada una de las unidades de modularización *aspecto* concentra y encapsula la implementación de cada uno de los *conceptos transversales*, buscando la minimización del acoplamiento, y consiguiendo extraer su tratamiento del código núcleo del sistema. En consecuencia, éste deja de embeber la lógica no funcional, consiguiendo aumentar la cohesión y minimizar el desacoplamiento.

Estos efectos reportan numerosos beneficios, como son: una mayor productividad, un mayor grado de legibilidad y facilidad de mantenimiento, facilidad de extensión del sistema y un mayor grado de reutilización. En definitiva, todo ello se traduce en una notable mejora de la calidad del software. Además, cabe mencionar, que mediante la aplicación de la POA se evita el denominado *Dilema del arquitecto* definido anteriormente (véase *sección 6.2.3.1.*), puesto que las decisiones relativas a qué requisitos abordar puede posponerse hasta el momento en que se esté en condiciones de tomar una decisión al respecto, ya que es posible implementar los futuros requisitos como *aspectos* separados. En definitiva, una implementación débilmente acoplada representa la clave para una elevada reutilización de código.

Todo esto puede ser sintetizado diciendo que se consigue que *"el espacio de implementación sea n-dimensional, al igual que el espacio de requisitos"*, y por consiguiente el mapeo requisitos-implementación obtenido es mucho más apropiado [Lad02]. En otras palabras, las propiedades que son ortogonales se mantienen como tal en la implementación basada en POA, donde cada requisito se representa en una dimensión distinta.

Otro de los principios fundamentales de la POA que la hacen especialmente atractiva es el hecho de que permite que la lógica funcional del sistema pueda ser aumentada añadiendo

requisitos extra-funcionales sin alterar el código, y sin que ni siquiera sea consciente de ello. Este aspecto, que otorga gran poder a la POA se conoce con el nombre de *principio de la inconsciencia* (del inglés *obliviousness principle*), introducido por Filman et al. en [FF04].

El *principio de inconsciencia* se hace factible por el hecho de que en POA el flujo de composición va de los conceptos transversales hacia la funcionalidad principal, mientras que en POO el flujo va en la dirección opuesta. POA y POO pueden y suelen coexistir, manteniendo de ese modo intactos los compromisos y responsabilidades del sistema base.

6.2.3.3.2. Proceso de construcción de un programa en POA

POA permite implementar conceptos individuales bajo un estilo de programación débilmente acoplado, combinando posteriormente cada una de estas implementaciones resueltas por separado para formar el sistema final. La unidad de modularización en POA es, como ya se ha dicho, la llamada *aspecto*.

El desarrollo de un programa en POA implica tres pasos distintos:

1. *Descomposición aspectual*. Consiste en identificar las competencias básicas (funcionalidad básica o lógica de la aplicación) y las competencias transversales (funcionalidad extra o secundaria) que se dispersan por el código. En el caso que nos ocupa esta distinción ya viene resuelta: los requisitos funcionales son todos los que atañen a la aplicación y los requisitos extra-funcionales son los que corresponden al tratamiento del contexto y la correspondiente adaptación de la aplicación.

2. *Implementación de conceptos*. Consiste en implementar cada concepto identificado por separado, combinando la POA con un cierto *lenguaje de procedimiento generalizado*[4] [KLM+97] (generalmente un lenguaje *orientado a objetos*), al que se le llama *lenguaje base*.

   Las competencias básicas serán implementadas utilizando los componentes asociados al *lenguaje base* (si es un lenguaje *orientado a objetos*, generalmente son *clases*) y las *competencias transversales* serán las que serán implementados utilizando *aspectos*.

[4] Lenguaje de programación en que sus mecanismos de abstracción y composición tienen una raíz común en forma de procedimiento generalizado. Dentro de esta categoría se encuentran los lenguajes orientados a objetos, procedurales y funcionales. Para un lenguaje orientado a objetos el procedimiento generalizado es una clase; para uno procedural lo es un procedimiento y para uno funcional una función. Este concepto es un indicativo de que la POA es aplicable a cualquier paradigma de programación, esto es, que no es combinable exclusivamente con la POO.

3. *Recomposición aspectual.* Este paso consiste en enlazar cada uno de los conceptos implementados (*clases* y *aspectos*) para obtener una unidad de programa compacta, de acuerdo a ciertos criterios que son también suministrados como parte de las unidades *aspecto* utilizadas. Estos criterios se denominan *reglas de recomposición*, que especifican la manera como realizar la composición de todos los conceptos implementados independientemente.

   Para llevar a cabo este paso se utiliza un integrador de *aspectos* (un compilador específico del lenguaje de POA utilizado), al que se le suele llamar 'tejedor', término procedente del inglés *weaver*. Este proceso, por analogía recibe el nombre de *weaving*, el cual está gobernado por las citadas *reglas de recomposición*.

   Es precisamente la manera como especificar estas reglas de composición (su sintaxis y sus posibilidades) lo que caracteriza la esencia de cada lenguaje de POA, y que en ocasiones esa sintaxis constituye una extensión a un lenguaje de POO ya existente, como es el caso de *AspectJ* para el lenguaje convencional Java.

La figura 6.1 refleja el proceso de construcción de un programa en POA.

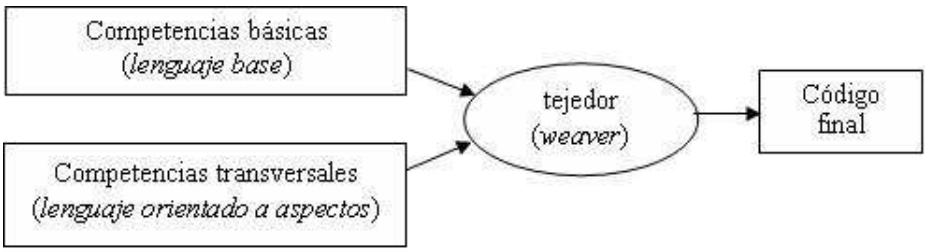


Figura 6.1: Esquema correspondiente al proceso de construcción de un programa en POA.

El proceso de *descomposición aspectual* aquí descrito suele representarse utilizando la siguiente analogía de la vida cotidiana: un prisma de cristal que deja pasar un haz de luz, mostrando los siete colores del arco iris. Esta metáfora refleja que el código de un aplicación normalmente es responsable de gran cantidad de requisitos que, para ser resueltos satisfactoriamente, conviene separar tal y como hace el prisma con el haz de luz (en este caso el identificador de conceptos). El haz de luz corresponde al código de la aplicación, los siete colores del arco iris corresponden a la descomposición en los distintos conceptos abordados por la aplicación o producto software y el prisma se asimila con el identificador de conceptos. Efectivamente, éste deja entrever las distintas responsabilidades que deben ser abordadas.

Este símil se refleja en la primera parte (izquierda) de la figura 6.2. La segunda parte representa el proceso de recomposición necesario para obtener el sistema final.

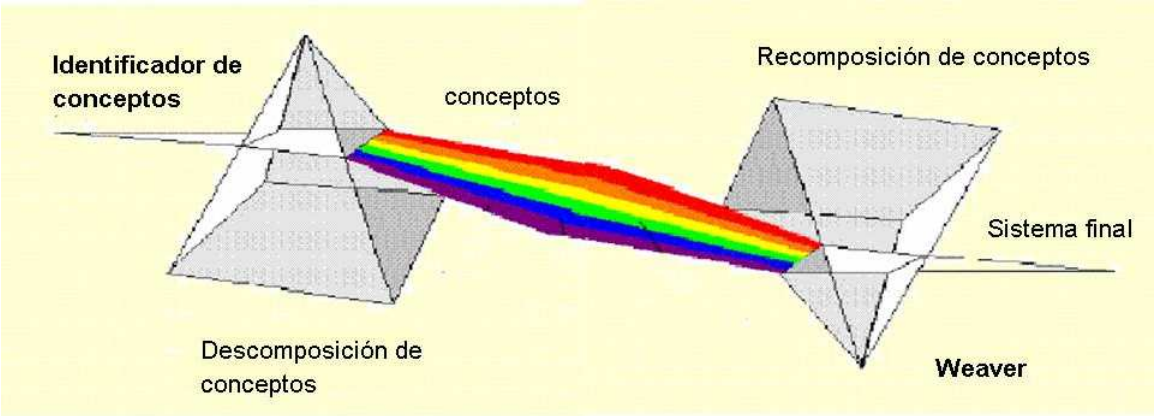


Figura 6.2: Analogía en el proceso de construcción de un programa en POA.

6.2.3.3.3. Reglas de recomposición propias de POA

Llegado este punto conviene introducir las denominadas *reglas de recomposición*, entendidas como un conjunto de reglas y criterios que se encargan de gestionar el despliegue del código encapsulado en las unidades *aspecto* en el resto de los componentes del sistema.

El concepto central alrededor del cual giran todas las acciones y funciones relacionadas con el paso de recomposición es el concepto de *punto de enlace* o también *punto de unión*. Se trata de un concepto fundamental de la POA que se define a continuación.

Definición 6.2 (Punto de enlace): Son los puntos en el flujo de ejecución de un programa perfectamente inidentificables y bien definidos correspondientes a los puntos en los que se desea incrementar la funcionalidad núcleo del sistema, con objeto de llevar a cabo el tratamiento del requisito extra-funcional en cuestión. Se dice que son los puntos donde será "inyectado" el código encapsulado en la unidad *aspecto*, alterando de ese modo el comportamiento subyacente. □

En efecto, a pesar de incrementar el nivel de abstracción, las competencias transversales no son totalmente independientes, y de alguna manera debe establecerse la relación que liga los componentes con los *aspectos*. En otras palabras, las distintas componentes (*aspectos* y clases) tienen que interactuar de alguna manera definiendo puntos de intersección. Esos puntos son los *puntos de enlace*, los cuales establecen el paso de recomposición o entretejido entre los mismos.

Se suele utilizar el término *modelo de puntos de unión* para referirse al conjunto de *puntos de unión* que se exponen en un sistema y cómo son capturados. Este tipo de interferencia sobre el código subyacente es el que se aplica de manera transparente, es decir, sin que el sistema subyacente conozca de la existencia de la entidad *aspecto* ni de los mecanismos asociados.

Ejemplos de *puntos de enlace* son el inicio de la ejecución de un método, su finalización, la llamada a un método o constructor, la inicialización o acceso para lectura o escritura de objetos o atributos, etc.

Cabe señalar que, en general, la identificación de los *puntos de enlace* adecuados para inyectar el comportamiento extra requiere un profundo conocimiento del código base. Como resultado, se genera un fuerte acoplamiento de los *aspectos* hacia las clases interferidas, lo cual va en detrimento del objetivo de alta reutilización planteado para el *Framework de Plasticidad Implícita*. Este inconveniente es el que se propone atacar combinando el uso de *aspectos* con *anotaciones de metadatos*, tal y como se expone en el *Capítulo 8* (véase *Capítulo 8*; *sección 8.2.1.1.*).

Relacionado con el *punto de enlace* se utiliza una construcción del lenguaje denominada *punto de corte*, o simplemente *corte*. El *punto de corte* permite agrupar y definir en una sola construcción del programa aquellos *puntos de unión* para los que se requiere aplicar un mismo comportamiento o tratamiento extra, a través de un mismo código. Los *puntos de corte* capturan los *puntos de unión* que hayan estado establecidos en tiempo de compilación a lo largo de la ejecución del programa, con objeto de inyectar ese código extra. Adicionalmente, son capaces también de capturar el contexto de ejecución de manera totalmente transparente, exponiéndolo por tanto a posibles manipulaciones.

El código que describe el comportamiento adicional a ser insertado en el código base se denomina *consejo* (también guía u orientación). El *consejo* constituye otra construcción que forma parte también de la entidad *aspecto*, y contiene el código a ejecutar cuando los *puntos de unión* son alcanzados y capturados por los correspondientes *puntos de corte* a lo largo de la ejecución del programa. Así, el *consejo*, que equivale al concepto de *método* en POO, corresponde al código responsable de manejar el requisito extra-funcional para el que está definido el *aspecto*, aumentando de ese modo la funcionalidad núcleo. Este código se expresa mediante construcciones propias de la POO, esto es, consistente en código Java puro.

En particular, las reglas de recomposición vienen especificadas por los *puntos de corte* y los *consejos*, los cuales van siempre emparejados.

Por último, la unidad de programa *aspecto* es la agrupación lógica de todos los *puntos de corte* y los *consejos* asociados, los cuales se encargan del tratamiento de un determinado

concepto o requisito extra-funcional, aquel para el que se construye la entidad *aspecto*. Es la unidad *aspecto* la construcción de programa que concentra y encapsula una determinada funcionalidad extra (un *concepto transversal*). El concepto de *aspecto* puede ser comparado al de una clase, así como también su estructura asociada.

6.2.3.3.4. Ventajas e inconvenientes

Por supuesto, no todo son ventajas en la adopción de la POA. A continuación se relacionan las ventajas más importantes, así como también algunos de sus inconvenientes.

Ventajas

A pesar de haber sido ya perfiladas la mayoría de las ventajas de la POA, a continuación se relacionan las más importantes.

- *Evita el Dilema del Arquitecto* (véase *sección 6.2.3.1.*), permitiendo posponer las decisiones de diseño.
- *Proporciona alta productividad y reducción de costes*, al permitir a cada miembro del equipo desarrollador centrarse en un requisito en particular. En muchas ocasiones también se manifiesta en una reducción en el tamaño del código.
- *Mayor modularidad* y, por lo tanto, mejor asignación de responsabilidades. En consecuencia, se obtiene un mayor grado de legibilidad y facilidad de mantenimiento, con el mínimo acoplamiento (dependencias entre módulos) y la máxima cohesión en cada módulo, principios básicos del diseño orientado a objetos.
- *Facilidad de extensión y evolución del sistema*. El *principio de la inconsciencia* es clave para la evolución del software.
- *Mayor grado de reutilización*, gracias a la minimización del acoplamiento.

Inconvenientes

La mayoría de las críticas recibidas han sido atribuidas a la curva de aprendizaje. No obstante, esto es algo inevitable propio de la adopción de una nueva filosofía de programación, y que sin duda ha tenido que afrontarse en cada nuevo paradigma de programación. En cualquier caso, la cuestión a valorar es: ¿Los beneficios esperados compensan el esfuerzo que tiene que hacer una empresa para cambiar su manera de programar? Eso seguramente dependa de la organización, así como de la trascendencia de los problemas de modularización de código con que se encuentre.

Otra de las críticas recibidas, muy relacionada con la anterior, ha sido la dificultad en la comprensión del funcionamiento del programa. En parte esta dificultad viene provocada por el hecho de que resulta difícil en muchas ocasiones seguir la pista de qué código de la aplicación está siendo afectado por qué *aspecto*. Las herramientas de visualización de *aspectos* y otras convenciones de codificación pueden simplificar esta dificultad.

Otro de los inconvenientes con los que cuenta es el hecho de que no existe un estándar para representar *aspectos* en UML, a pesar de existir algunos intentos al respecto. Sin una notación estándar disponible, la coordinación dentro del equipo de desarrollado se complica.

Por otro lado, la habilidad de violar los principios de encapsulamiento y ocultación de datos, los cuales constituyen principios básicos del diseño orientado a objetos constituye otro foco de crítica. En un informe técnico de la Universidad de Virginia [HLM99] se establece que muchos de los principios centrales a la POO son ignorados en el diseño orientado a *aspectos*. Sin duda, esta es una de las desventajas de los lenguajes orientados a *aspectos* actuales. Sería deseable proporcionar una interfaz para que *aspectos* y objetos se comunicaran respetando las restricciones de encapsulamiento de código [LAC05].

No obstante, bajo la opinión del autor de esta tesis, el mayor inconveniente de la POA es el problema conocido como *interferencia* entre *aspectos* o también como *interacciones* o *conflictos* entre *aspectos* [DFS02]. Tanto es así que este tópico se está convirtiendo en una nueva área de investigación activa, tal y como se analiza en [ZVG06]. Una situación de *interferencia entre aspectos* se produce cuando un sistema incluye múltiples *aspectos* y varios de ellos interceptan un mismo punto en el flujo de ejecución del programa (un mismo *punto de enlace*). Si llegado a ese momento ambos *puntos de enlace* cumplen todos los requisitos para ser activados, entonces entran en conflicto. En esos casos debe establecerse algún mecanismo de resolución de conflictos para evitar comportamientos imprevistos.

En determinados casos es suficiente con establecer el orden de aplicación de cada uno de ellos, estrategia conocida como reglas de *precedencia entre aspectos*, pero otras veces puede ser necesario establecer mecanismos más complejos. La diferencia entre ambos casos depende de si esas interacciones pueden ser o no detectadas de antemano, o bien dependen de las condiciones de ejecución, y por lo tanto no pueden ser establecidas mediante una construcción sintáctica, esto es, en tiempo de compilación, sino que se requieren nuevos enfoques. Por otro lado, tal y como se analiza en [ZVG06], los conflictos entre *aspectos* pueden darse incluso si éstos no están afectando el mismo *punto de enlace*. En particular, el enfoque propuesto en [ZVG06] es novedoso por el hecho de centrarse en la semántica y comportamiento de los *aspectos*, en lugar de hacerlo desde el punto de vista sintáctico.

El lenguaje *AspectJ* proporciona un mecanismo de *precedencia entre aspectos* explícito muy sencillo que resuelve los posibles conflictos en tiempo de compilación.

6.2.4. Justificación de la adopción de POA

Una vez analizada esta filosofía de programación en detalle, y de haberla distinguido como la más apropiada entre las distintas tecnologías de *separación de conceptos*, cabe plantearse la idoneidad de su aplicación en el campo objeto de estudio: el desarrollo de sistemas software *sensibles al contexto*, una vez desarrollada toda la funcionalidad núcleo.

6.2.4.1. Idoneidad con respecto al campo de aplicación

Ya han sido expuestos anteriormente los objetivos perseguidos mediante la construcción del *Framework de Plasticidad Implícita* desarrollado en esta tesis. Su aplicación permitirá que una aplicación que no es *sensible al contexto*, generalmente una aplicación móvil, se comporte como tal, una vez instanciados y acoplados los componentes del framework especializados para una aplicación y unas necesidades contextuales determinadas. Se plantea como un medio para hacer aumentar los sistemas con funcionalidad extra relativa al tratamiento del contexto y consecuente adaptación de la aplicación, siendo ésta exclusivamente la responsabilidad de los componentes del framework.

6.2.4.1.1. Mecanismos de adaptación relativos al contexto

En primer lugar, la cuestión más importante para demostrar la idoneidad de la técnica elegida es que, tal y como se preveía desde un principio y constituye una de las hipótesis planteadas en esta tesis (véase *Hipótesis 3 - Capítulo1*; *sección 1.3.*), efectivamente el tratamiento de las *restricciones de tiempo real*, así como los mecanismos de adaptación para ajustar la IU a las *variaciones dinámicas en el contexto de uso*, constituyen *conceptos transversales* (véase *Definición 6.1*).

En efecto, mediante la utilización de las técnicas de *orientación a objeto*s, el código encargado de dotar al sistema de la capacidad de ser *sensible al contexto* aparece inevitablemente diseminado en forma de código repetido a lo largo de la mayoría de los módulos del sistema. En consecuencia, la funcionalidad núcleo queda diluida al tener que tratar también con toda esta funcionalidad. El resultado es un código enmarañado que provoca los consecuentes efectos negativos ya comentados en la legibilidad y mantenimiento del código, como consecuencia del tratamiento de los *conceptos transversales* únicamente con objetos. En consecuencia, la ortogonalidad entre las distintas *restricciones de tiempo real* es nula, lo que significa que se complica enormemente la construcción de distintas versiones

del sistema apropiadas para distintas necesidades contextuales, dificultando su flexibilidad y reutilización. Cabe destacar también que cuanto mayor es el número de *restricciones de tiempo real* a tratar, mayor es el nivel de contaminación de código.

Para demostrar esta premisa, que confirma la *Hipótesis 3* (véase *Capítulo 1*; *sección 1.3.*), se han realizado una serie de pruebas experimentales destinadas a contrastar y cuantificar tanto la magnitud como el efecto de esa dispersión y enmarañamiento de código utilizando distintas versiones de un mismo caso de estudio utilizando tan sólo POO o bien POO en combinación con POA. Los detalles acerca de la experimentación llevada a cabo (la aplicación utilizada en las pruebas, los prototipos diseñados, los atributos del contexto estudiados y, por último, los resultados obtenidos) se desarrollan en el *Capítulo 8*.

Se adopta, por tanto, la POA como técnica de separación de conceptos más apropiada en estos casos, tal y como ya se discute en la *sección 6.2.2.4*. El objetivo de su implantación consiste en extraer y encapsular en una sola construcción del programa (en una unidad de modularización) todo el tratamiento relativo a cada *restricción de tiempo real*, evitando su dispersión.

En segundo lugar, el acoplamiento de esa nueva funcionalidad, encapsulada en *aspectos*, en el sistema de partida debe hacerse de manera que éste permanezca ajeno a la incorporación de los nuevos módulos, los cuales han estado diseñados e implementados en independencia de la funcionalidad y código base del sistema destinatario. Este paso evita tener que recodificar el sistema de partida, proceso que requeriría un esfuerzo muy importante de refactorización.

A pesar de que, como ya se ha comentado, la integración de *aspectos* en un sistema ya existente utilizando POA le resulte transparente a aquel (*principio de inconsciencia*), eso no significa que el paso de acoplamiento sea sencillo ni directo. Requiere un esfuerzo importante por conocer el código y propiedades principales del sistema base. Ese es uno de los principales inconvenientes ya mencionados anteriormente en el planteamiento de las premisas iniciales (véase *sección 6.2.1.2.1.*).

En este aspecto, cabe destacar que los puntos de la ejecución en los que debe incorporarse la funcionalidad extra (los *puntos de enlace*), para el caso que nos ocupa (adaptación de la IU y ciertos aspectos del comportamiento de la aplicación de acuerdo al contexto recogido durante la ejecución) involucran mayoritariamente a (1) las clases pertenecientes a la interfaz; (2) ciertas entidades clave relacionadas con la semántica de la aplicación en la que se desea incidir; y, dependiendo de los objetivos y tipo de componente (3) la ejecución de la funcionalidad concreta en la que se desea incidir, que en cualquier caso se trata de una funcionalidad puntual; o bien (4) ciertas acciones relacionadas con la coordinación

o interacción entre los distintos miembros del equipo de trabajo, en el caso de entornos colaborativos.

Es evidente que el esfuerzo necesario para adquirir todo o parte de este conocimiento –dependiendo de los componentes que se deseen integrar- no es trivial, y requiere profundizar en el código y estructura del sistema. No obstante, en cualquier caso, involucran aspectos muy puntuales de la lógica de la aplicación, íntimamente relacionadas con las adaptaciones a obtener o con el manejo de la IU. De hecho, el esfuerzo que supone conocer con precisión los objetivos que se pretenden conseguir mediante la nueva funcionalidad relativa al contexto se equipara, en cierto modo, con el esfuerzo requerido para conocer cómo llevar a cabo su instanciación, pudiéndose asimilar como un solo paso. En cualquier caso los *métodos* (pensando en la implementación de la aplicación base bajo el enfoque de la POO) involucrados son métodos especializados en acciones muy concretas, y que por tanto son fácilmente identificables como *puntos de enlace* en los que incidir. En otras ocasiones se trata de los métodos más significativos, generalmente comunes a la mayoría de aplicaciones, relacionados con el tratamiento de la IU. Por lo tanto, el conocimiento requerido para la integración del tratamiento del contexto se queda en un nivel superficial.

Bajo la opinión del autor de esta tesis, este esfuerzo resulta compensatorio en relación al que hubiera tenido que realizarse de tener que desarrollar la aplicación con toda su funcionalidad desde un principio, y por supuesto es considerablemente inferior al esfuerzo de recodificación que tendría que llevarse a cabo al decidir agregar la funcionalidad del contexto una vez completada la aplicación, de no disponer de un framework.

En tercer lugar, se debe tender hacia mecanismos lo más desacoplados posible para establecer los correspondientes *puntos de enlace* (véase *sección 6.2.3.3.3.*), a fin de evitar dependencias con el sistema. En la actualidad el manejo de las *anotaciones de metadatos* constituye ya una parte integrante de J2SE desde su versión 5.0 [Lad05]. Esta es la técnica adoptada para obtener este desacoplamiento. Aunque las *anotaciones de metadatos* no se encuentran todavía implementadas para la modalidad J2ME [Pir02] para móviles, se espera que lo esté en un futuro cercano. En ese momento la adaptación del framework actual a esa nueva característica será directa.

Por último, otro de los inconvenientes identificados en la *sección 6.2.3.3.4.* es la curva de aprendizaje requerida por parte del desarrollador de *aplicaciones sensibles al contexto*, al tener que afrontar un nuevo paradigma de programación. Tal y como se expone en los capítulos sucesivos, el diseño del framework se ha dirigido tratando de minimizar el código a incluir en las unidades *aspecto*, recurriendo también al uso de clases convencionales. Aún en el caso de tener que especializar los *aspectos* del framework, la parte en concreto a

personalizar es mayoritariamente la parte del código operativo, esto es, basado en construcciones puramente *orientadas a objetos*, tales como *métodos*. Para lograr este objetivo se vale de una tercera capa denominada *capa sensible al contexto*, la cual encapsula la mayor parte de la funcionalidad, tal y como se presenta en la *sección 6.3*. De ese modo se reduce enormemente la dificultad en la aplicación del framework.

6.2.4.1.2. Aptitud para dispositivos compactos

El uso de la POA puede ser aplicado a cualquier tipo de plataforma de sobremesa, plataforma Web o dispositivo compacto, a pesar de no existir todavía una versión optimizada de la librería de *AspectJ* para la modalidad J2ME, tal y como se explica en el *Capítulo 7* (*Capítulo* 7; *sección 7.4.2.2.2.*).

El hecho de programar y enlazar componentes adicionales a una aplicación compacta, juntamente con la librería necesaria para trabajar con *AspectJ*[5] planteó inicialmente la duda de si podría ser soportada y ejecutada por un dispositivo de capacidades limitadas. No obstante, esa duda ha sido despejada como consecuencia del éxito obtenido en las pruebas llevadas a cabo, a pesar de que el tamaño ejecutable (el .jar) resulta considerable. Eso refuerza tanto el requisito de *plasticidad* como el de *ubicuidad*, o dilución de la computación en pequeños aparatos, capaces de reaccionar al entorno que los rodea, así como de comunicarse con otros.

La POA es una tecnología relativamente nueva, y es habitual que para desarrollar una versión adecuada para dispositivos compactos en primer lugar deba haberse demostrado sobradamente su aceptación en el desarrollo de sistemas para las plataformas habituales, antes de adentrarse también en el mundo de los dispositivos compactos.

En cualquier caso, esta experimentación ha servido para demostrar que, efectivamente, la POA constituye un enfoque apropiado incluso para dispositivos compactos con recursos limitados, permitiendo la construcción de componentes de bajo consumo de recursos, los cuales cumplen con las expectativas inicialmente planteadas. Esto nos lleva a asentar su idoneidad para entornos ubicuos.

Una exposición detallada de la experimentación llevada a cabo se incluye en el Capítulo 7.

6.2.4.2. Satisfacción de las propiedades exigidas en cuanto al diseño

De los experimentos realizados a lo largo de la programación de los distintos prototipos *sensibles al contexto* desarrollados utilizando POA (véase *Capítulo 7*) se deduce que

[5] aspectjrt.jar

se satisfacen también las propiedades y requisitos de diseño inicialmente planteados. A continuación se discuten en detalle.

6.2.4.2.1. Transparencia

El hecho de que en efecto, el sistema subyacente no sufre alteración alguna por el hecho de integrarle funcionalidad extra, y ni siquiera es consciente de que ha sido enlazado con componentes adicionales, demuestra que este enfoque es efectivamente no invasivo, o lo que es lo mismo, que se alcanza una total transparencia, una de las propiedades exigidas tal y como ha quedado enunciada en la *sección 6.2.1.2.2*.

En el *Capítulo 7* (véase *Capítulo 7*; *sección 7.4.*) se dan detalles en relación a este aspecto.

6.2.4.2.2. Reutilización

La meta de transparencia alcanzada con el uso de la POA, teniendo en cuenta que lo que se pretende es separar en unidades de modularización separadas el tratamiento de cada *restricción de tiempo real*, repercute directamente en favor de la capacidad para hacer evolucionar el sistema y flexibilizar sus capacidades de adaptación, las cuales podrán ser acomodadas a distintas circunstancias, simplemente desarrollando el módulo pertinente y enlazándolo oportunamente a la aplicación base.

Esto se traduce en la propiedad de reutilizar fácilmente no sólo el sistema a distintas situaciones contextuales, sino también las propias capacidades de adaptación a distintos sistemas, siempre y cuando las dependencias con el mismo estén bien identificadas, encapsuladas y abstraídas.

6.2.4.2.3. Ortogonalidad

Respecto a la independencia de los componentes *sensibles al contexto*, cabe diferenciar entre la independencia con respecto a otras componentes encargadas del tratamiento de otro tipo de *restricciones de tiempo real*, y la independencia con respecto al sistema base. En el primer caso la independencia es total, puesto que se implementan por separado, utilizando para ello unidades de modularización distintas totalmente desacopladas entre sí.

En relación a las dependencias con respecto al sistema base cabe señalar que, por propia definición de la filosofía *orientada a aspectos*, tal y como ya se ha comentado, el sistema base no es consciente de esas nuevas funcionalidades, y por lo tanto no mantiene

dependencias con los *aspectos*. No obstante, la especificación de los *puntos de enlace* requiere un profundo conocimiento de la aplicación subyacente y, en consecuencia, introduce un elevado nivel de dependencia de la parte *aspectual* hacia las clases de la funcionalidad núcleo. Esto implica que los componentes encargados de la funcionalidad extra, por defecto, no pueden ofrecer el nivel de genericidad que se pretende.

Con el propósito de paliar este problema, y con la meta siempre puesta hacia la obtención de un framework genérico, se proponen dos medidas: (1) la combinación del uso de *aspectos* con *anotaciones de metadatos*, enfoque que se presenta en detalle en el *Capítulo 8* (véase *Capítulo 8*; *sección 8.2.1.1.*); y (2) una medida propia del diseño de cualquier tipo de framework, consistente en identificar y encapsular la especificación de esos puntos de "enganche" con la aplicación, de modo que estén localizados y concentrados también en un módulo concreto.

Una vez analizadas cada una de las facetas tanto a nivel de diseño software como a nivel de su idoneidad en el campo de aplicación, y habiendo sido contrastado con los enfoques más importantes de *Separación de Conceptos* (véase *sección 6.2.2.4.*), se llega a la conclusión de que, efectivamente, y tal y como se preveía desde un principio (véase *Hipótesis 4*; *Capítulo 1 – sección 1.3.*), POA favorece la integración de los mecanismos de adaptación proactiva en la operativa del sistema bajo unos cánones de calidad, destacándose como una de las técnicas más apropiadas. En consecuencia, el desarrollo de un framework genérico de soporte a esa integración resulta prometedor.

6.2.5. Lenguaje de POA seleccionado

AspectJ[6] [Xer00] constituye la principal implementación del paradigma de POA para Java y también una de las más populares y maduras, al contar con una gran comunidad de usuarios y desarrolladores [LAC05]. Es un lenguaje de propósito general.

Se trata de una implementación relativamente fácil de aprender por parte de los desarrolladores en Java, puesto que aporta un conjunto reducido de extensiones al lenguaje, a fin de establecer las reglas de recomposición y definir la construcción *aspecto*. El resto de código del *aspecto*, esto es, el que implementa su funcionalidad, el cual se concentra en la construcción denominada *consejo*, sigue siendo Java puro. Este es un aspecto relevante, en cuanto a que la dificultad que supone llevar el framework a la práctica se ve considerablemente reducida.

En consecuencia, el compilador de *AspectJ* es una extensión del compilador de Java convencional, el cual integra los *aspectos* en el código base, de acuerdo a la reglas de

[6]The AspectJ project. http://www.eclipse.org/aspectj/

recomposición, y a partir de aquí invoca al compilador de Java para generar bytecodes (el código intermedio) que, al ser estándar posibilita su interpretación por parte de cualquier máquina virtual de Java. Es por ello que la compatibilidad observada entre *AspectJ* y J2ME no resulta sorprendente.

Así, un programa en *AspectJ* puede ser considerado como un programa en Java convencional al que se le han añadido un conjunto de *aspectos*. En definitiva, la combinación de *AspectJ* con Java permite añadir las posibilidades de la POA a la vez que mantiene las características ventajosas de Java.

Del mismo modo, un *aspecto* en *AspectJ* puede considerarse como una clase con la particularidad de que contienen unos *constructores de corte* que no existen en Java. Los *cortes* de *AspectJ* capturan colecciones de eventos en la ejecución de un programa. Estos eventos pueden ser invocaciones de métodos, invocaciones de constructores, y excepciones de señales y gestión. Por lo tanto, un *aspecto* puede afectar a la implementación de un número de métodos en un número de clases, lo que permite cortar de forma transversal la modularidad de las clases de forma limpia y cuidadosamente diseñada [Rei00].

La primera versión fue lanzada en 1998 por el equipo liderado por Gregor Kiczales [KHH+01a]. Actualmente, y desde el 2002 en que fue transferido por Xerox PARC, pertenece a la comunidad de código abierto, integrado en *Eclipse Foundation*. Su distribución integra herramientas propias, y al mismo tiempo es integrable en la mayoría de entornos de desarrollo integrados. Se trata de un lenguaje relativamente nuevo que se encuentra en continua evolución.

En particular, la versión con la que se han realizado todas las experimentaciones a lo largo de la realización de la presente tesis es la versión 5.0[7] [Col05].

En el *Apéndice A* se presentan detalles acerca de las construcciones propias del lenguaje y otros aspectos generales de la POA en *AspectJ*.

6.3. Arquitectura para el *Motor de Plasticidad Implícita*

A continuación se describe la arquitectura adoptada para el *Motor de Plasticidad Implícita*, cuya estructura es el resultado de la consideración de todas las premisas iniciales y requisitos de diseño planteados desde su concepción. Por supuesto, su disposición final y otro tipo de detalles son consecuencia de las decisiones de implementación tomadas al respecto, las cuales han sido analizadas y justificadas anteriormente.

[7] The AspectJTM 5 Development Kit Developer's Notebook.
http://www.eclipse.org/aspectj/doc/next/adk15notebook/index.html

Cabe señalar que con objeto de maximizar la flexibilidad, y por lo tanto la reutilización del *Framework de Plasticidad Implícita*, el código de las unidades *aspecto* debe ser lo más conciso posible, enfocándose únicamente en las tareas de introspección, monitorización e integración de código extra. El resto de la lógica relativa al tratamiento del contexto es preferible encapsularla en otra capa distinta. Es por ello que se incorpora la denominada *capa sensible al contexto*. El código de los *aspectos* se limita por tanto a coordinar y enlazar la funcionalidad núcleo con la nueva funcionalidad del contexto, gobernando toda la lógica en su conjunto. Esta perspectiva proporciona no sólo el desacoplamiento perseguido entre los componentes del framework, sino también entre las funcionalidades núcleo y *sensible al contexto*, residentes respectivamente en la aplicación y en la *capa sensible al contexto* del *Motor de Plasticidad Implícita*.

En esta sección, en primer lugar se da una explicación elemental de cada una de las capas que constituyen la arquitectura del *Motor de Plasticidad Implícita*. A continuación se detallan las consideraciones específicas acerca de la sensibilidad al desarrollo de la actividad grupal, propias de entornos colaborativos, a fin de construir *IUs sensibles al grupo* (véase *Definición 4.1*; *Capítulo 4 - sección 4.1.*). La descripción de la estructura se realiza teniendo en cuenta el producto final, esto es, una vez se han instanciado los componentes relativos al contexto en la aplicación destino, con objeto de dotarla de *sensibilidad al contexto*.

6.3.1. Descripción por capas

Un *Motor de Plasticidad Implícita* se estructura en forma de arquitectura software dividida en tres capas, las cuales se disponen en el siguiente orden: *capa lógica*, *capa aspectual* y *capa sensible al contexto*. A continuación se presentan cada una de ellas.

6.3.1.1. Capa lógica

La *capa lógica* es la que contiene el sistema base de partida, esto es, una aplicación concreta con todas las entidades características de su dominio. Se trata de la funcionalidad núcleo, la cual está habitualmente formada tanto por los componentes de la lógica de negocio, como por los módulos encargados del acceso a los datos del dominio y la implementación de la interfaz. En cualquier caso no se incluye ningún tipo de restricción ni tratamiento relativos al contexto.

Tan sólo debe ser estudiada con más o menos detalle en el paso de acoplamiento de los componentes del framework, como paso necesario para su instanciación, a fin de establecer adecuadamente los *puntos de enlace*.

Esta capa constituye la capa inferior teniendo en cuenta el nivel de abstracción.

6.3.1.2. Capa sensible al contexto

Es la capa que contiene los componentes necesarios para representar y almacenar la información contextual, con objeto de integrar dicho contexto en el funcionamiento del sistema. A partir de ahora para hacer referencia a esa representación se utiliza el término *modelo contextual*, entendiéndolo como una materialización de los datos del contexto recopilados, a fin de ser manejados a nivel de programa.

Su encapsulación en una capa separada facilita su evolución y promueve la independencia. Además, simplifica en la medida de lo posible la complejidad de la *capa aspectua*l, al liberarla de la mayor parte del código relativo a la adaptación. Esto reporta una ventaja importante a la hora de comprender e instanciar el *Framework de Plasticidad Implícita* por parte del desarrollador de *aplicaciones sensibles al contexto*, liberándolo en cierto modo de tener que conocer en profundidad el paradigma de la POA en general y, en particular, las construcciones del lenguaje *AspectJ*.

Por lo que respecta a la parte relativa a la captura del contexto, en ocasiones es la *capa sensible al contexto* la responsable, y en otros casos es la *capa aspectual*, distinguiéndose dos patrones distintos en función de la naturaleza de los distintos atributos que caracterizan el contexto. Los atributos que integran la información contextual, según la caracterización realizada en la presente tesis se presentan en el *Capítulo 2* (véase *Capítulo 2*; *sección 2.1.3.*).

Así, tanto los atributos relativos al usuario como los relativos al grupo de trabajo, en el caso de actividades colaborativas, requieren una inspección y monitorización constante de la evolución y seguimiento del comportamiento del usuario o, en su caso, de la actividad grupal durante la ejecución. En decir, sólo es posible inferir las preferencias del usuario y la *consciencia de grupo particular* a partir de la observación de la propia evolución de la ejecución. En estos casos la adquisición del contexto es 'introspectiva' (véase definición en el *Capítulo 4*; *sección 4.1.4.1.4.*), y por lo tanto su codificación está fuertemente ligada a la lógica de la aplicación. Esta responsabilidad se delega en la *capa aspectual*, no sólo porque dispone de los mecanismos adecuados de monitorización e interferencia en la lógica del programa, sino también porque de ese modo se consigue desacoplar esta operativa de las otras dos capas: la *capa lógica* y *sensible al contexto*, y también evita el acoplamiento entre estas dos capas. Éste es, tal y como se expone a continuación, uno de los objetivos de la *capa aspectual*.

Sin embargo, en los componentes encargados del tratamiento de las restricciones relativas al entorno físico y los recursos hardware los datos del contexto asociados provienen del mundo exterior o de la inspección del propio equipo de computación. Por consiguiente, no se requiere realizar ningún tipo de seguimiento de la lógica de la aplicación. En definitiva, la adquisición del contexto constituye una operación totalmente desacoplada de la *capa lógica*, siendo en estos casos la *capa sensible al contexto* la responsable de llevarla a cabo. Cabe recordar que la *capa aspectual* debe ser lo más simple posible, fomentando la *"separación de la adquisición del contexto, del uso que se hace del mismo"* (véase el requisito número 1 en el *Capítulo 5*; *sección 5.2.2.*).

Los detalles al respecto de estos dos patrones relativos a la adquisición del contexto se explican en detalle y para cada caso en el *Capítulo 7* (véase *Capítulo 7*; *sección 7.3.1.2.*).

En el caso particular de sistemas colaborativos esta capa incluye también la información relacionada con la *consciencia de grupo particular* (véase definición en el *Capítulo 2*; *sección 2.1.3.*). Principalmente, esta componente reúne los datos relacionados con las interacciones usuario a usuario, los eventos de comunicación y las acciones de coordinación que representan el entendimiento que cada miembro tiene al respecto del desarrollo de la actividad grupal. Tal y como se comenta en la *sección 6.3.2.*, el *Motor de Plasticidad Implícita* se responsabiliza de mantener esta información actualizada.

Por último, cabe señalar que la *capa sensible al contexto* debe ser lo suficientemente flexible como para expresar los distintos elementos que definen el entorno contextual de la aplicación, tal y como hayan sido establecidos en la fase de diseño. Es por ello que estará integrada por tantas componentes como sean necesarias para dar tratamiento a todos los atributos que hayan sido establecidos, preservando en todo momento la ortogonalidad entre ellos. Esta capa constituye la capa superior.

6.3.1.3. Capa aspectual

Esta capa tiene la función de encapsular, gobernar y dirigir todos los pasos y decisiones relacionados con la ejecución de la adaptación propiamente dicha de la *capa lógica* -la aplicación subyacente-, en los términos que hayan sido establecidos en la etapa de diseño, y de acuerdo al estado de la información contextual almacenada en la *capa sensible al contexto*.

Dependiendo de los casos, tal y como se ha mencionado anteriormente, la *capa aspectual* puede incluir también los mecanismos necesarios para la captura de la información contextual, a fin de ser almacenada en su correspondiente representación (el *modelo contextual*). Entre las decisiones más importantes canalizadas por la *capa aspectual* está la

de determinar cuándo las *variaciones dinámicas en el contexto de uso* detectadas han de dar lugar al desencadenamiento de los mecanismos de adaptación para los que ha estado concebido el *Motor de Plasticidad Implícita* en cuestión.

Esta capa perfectamente podría denominarse *capa adaptativa*. No obstante, por el hecho de ser la POA la tecnología seleccionada para su implementación, se ha preferido adoptar el nombre de *capa aspectual*.

Los objetivos de esta capa se pueden resumir en los dos siguientes: (1) proporcionar los mecanismos adecuados para el seguimiento, interferencia y alteración de la lógica del programa, así como de integración de nueva funcionalidad utilizando para ello las posibilidades de la POA; y (2) proporcionar el vínculo necesario entre las otras dos capas, con el propósito de alcanzar el máximo desacoplamiento entre ambas. Es por ello que se concibe como la capa intermedia que sirve de enlace transparente entre las *capas lógica* y *sensible al contexto*.

En efecto, la *capa aspectual* constituye el vehículo que hace posible reflejar el estado en curso de los atributos del contexto, modelados a través del *modelo contextual*, en la lógica e IU de la aplicación sin que las capas implicadas sean conscientes de su coexistencia. Esta peculiaridad preserva la ortogonalidad entre las *capas lógica* y *sensible al contexto*, y por consiguiente su reutilización. En particular, de este modo se consigue que la *capa sensible al contexto* sea independiente del dominio de aplicación y la lógica de negocio, mientras que la *capa lógica* puede evolucionar en independencia de las demás, permaneciendo intacta a pesar de la integración de la nueva funcionalidad. Esta ortogonalidad se consigue canalizando y concentrando todas las dependencias entre las capas extremas en la *capa aspectual*, y es con este objetivo que debe ser implementada.

En conclusión, la *capa aspectual* constituye el núcleo del *Motor de Plasticidad Implícita*, al ser la parte que activa la funcionalidad extra (en este caso relativa al contexto) en los puntos clave de la ejecución de la aplicación subyacente, constituyendo por tanto el brazo ejecutor del *Motor de Plasticidad Implícita*.

En el caso particular de sistemas colaborativos, la misión principal de esta capa es la de encapsular y poner en funcionamiento los *mecanismos locales de consciencia de grupo*, de acuerdo a la *consciencia de grupo particular* almacenada en la capa superior. Es la *capa aspectual* la responsable de supervisar, manipular y actualizar la *consciencia de grupo particular* propia de cada individuo.

6.3.1.4. Representación gráfica de la arquitectura

La figura 6.3 muestra el esquema de la arquitectura dividida en capas propuesta. La capa inferior es la que contiene la aplicación base que, por supuesto, está formada por los distintas componentes que ponen en funcionamiento la operativa central de la aplicación.

Los detalles de cada una de ellas, en particular de las dos superiores, que son las que integran el *Framework de Plasticidad Implícita* se proporcionan en el *Capítulo 8*, destinado a su presentación.

El estereotipo "*crosscuts*" se refiere a la acción de interceptar la lógica funcional y/o integrar el código del *aspecto* en la aplicación base. Como ya se ha mencionado en la *sección 6.2.3.3.4*, no existe ningún estándar al respecto.

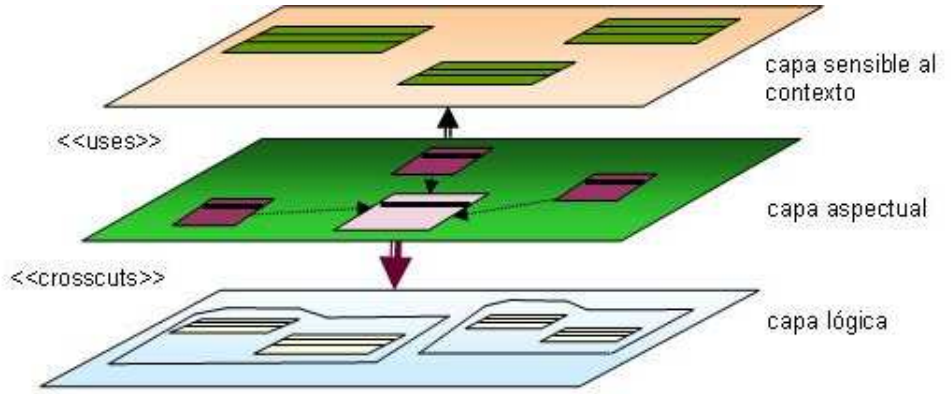


Figura 6.3: Esquema de la arquitectura software para el *Motor de Plasticidad Implícita*

6.3.2. Consideraciones específicas acerca de la colaboración

En el tratamiento de entornos colaborativos, la misión del *Motor de Plasticidad Implícita* se ve incrementada puesto que adicionalmente se encarga de capturar y mantener la *consciencia de grupo particular* (véase definición en la *sección 2.1.3.*), poniendo en funcionamiento algún tipo de *mecanismo local de consciencia de grupo* (véase *Definición 2.2*; *Capítulo 2 - sección 2.1.3.*), con el propósito de fomentar la interacción usuario a usuario entre los componentes del grupo de trabajo. En estos casos la *consciencia de grupo particular* constituye una parte fundamental del *modelo contextual*.

Para ser más precisos, los objetivos de un *Motor de Plasticidad Implícita* específicamente diseñado para entornos colaborativos son los dos siguientes:

1. monitorizar y hacer un seguimiento del estado de la actividad grupal desde una perspectiva individual, así como también de las interacciones del usuario con los otros miembros integrantes del grupo de trabajo. Estos son los aspectos fundamentales que conforman la información de la *consciencia de grupo particular*.

2. disparar en segundo plano ciertas acciones de coordinación y eventos de comunicación, o bien de aplicar pequeños reajustes en las tareas inmediatas en favor de la colaboración, de la forma más adecuada posible a la situación de la propia actividad, y bajo las circunstancias que hayan sido establecidas de antemano. Esto se consigue haciendo uso de los mecanismos propios de la POA. De esto es de lo que se encargan los *mecanismos locales de consciencia de grupo*, los cuales aparecen encapsulados en los *aspectos*.

Por lo que respecta a la *consciencia de grupo particular*, ésta es alimentada por los *mecanismos locales de consciencia de grupo*, los cuales, a su vez, actúan de acuerdo al estado de aquélla, constituyendo un ciclo de realimentación constante. Por lo tanto, la *consciencia de grupo particular* responde a una doble misión:

(a) servir de directriz para la actuación de los *mecanismos locales de consciencia de grupo*, cuyas responsabilidades ya han quedado expuestas arriba.

(b) contribuir a la construcción del *conocimiento compartido* [CGPO02] (véase *Capítulo 2*; *sección 2.1.3*) relativo a la situación del grupo de trabajo y el estado de la actividad grupal. Cabe recordar que el *conocimiento compartido* se construye como resultado de reunir toda la información individual de cada miembro del equipo de trabajo. Así, la *consciencia de grupo particular* forma parte de la información a ser transmitida y compartida con el resto de los componentes del grupo de trabajo. En el *Capítulo 4* se comentan los beneficios de disponer de un *conocimiento compartido* (véase *Capítulo 4*; *sección 4.1.2.3.*).

En efecto, la *consciencia de grupo particular* proporciona una visión particular de la actividad grupal que contribuye en la toma de decisiones locales relativas al grupo de trabajo y al desarrollo de la actividad grupal, desencadenando acciones extra-funcionales concretas. No obstante, el mantenimiento de la *consciencia de grupo particular* adquiere verdadero sentido cuando se introducen los mecanismos necesarios para coleccionar las distintas perspectivas de cada uno de los miembros del grupo de trabajo, conformando el denominado *conocimiento compartido*. El *conocimiento compartido* se reúne en el *servidor de plasticidad*, responsable de ofrecer una perspectiva global de la actividad grupal, tal y como se menciona en el *Capítulo 4*. Sólo de ese modo se consigue tomar en consideración una perspectiva conjunta del estado de la actividad grupal, necesaria para conseguir una verdadera colaboración entre los distintos integrantes del grupo.

En definitiva, se combinan dos perspectivas distintas relativas al grupo: (1) una perspectiva individual, distribuida entre todos los miembros, que es la que se construye a través del *Motor de Plasticidad Implícita*; y (2) una perspectiva global centralizada en el *servidor de plasticidad* y representada a través del *conocimiento compartido*. Estas dos perspectivas se equiparan con los dos niveles de plasticidad fomentados por la *visión dicotómica de plasticidad* propuesta en la presente tesis. Por otro lado, el mecanismo de propagación proporcionado por una infraestructura basada en la *visión dicotómica* constituye un mecanismo válido para difundir un entendimiento compartido, o *consciencia de conocimiento compartido* [CGPO02] (definida en el *Capítulo 2*; *sección 2.1.3*). Este enfoque responde a las premisas defendidas por los desarrolladores de groupware, los cuales defienden que en un entorno colaborativo deben ser proporcionados *mecanismos de consciencia de grupo* con el propósito de compartir cada perspectiva particular con el resto de los integrantes, a fin de asegurar la interacción entre ellos y alcanzar un entendimiento compartido [GG02].

La manera como el servidor hace llegar esa visión de conjunto es a través de la construcción y entrega de IUs convenientemente personalizadas a las condiciones del grupo de trabajo y al estado de la actividad grupal en curso. Este tipo de IUs son las que se califican en esta tesis como *IUs sensibles al grupo* (véase *Definición 4.1*; *Capítulo 4 - sección 4.1.*). De ese modo, el contexto global de la actividad grupal puede llegar a ser asimilado por todos los miembros del grupo, los cuales pueden, en consecuencia, trabajar de manera colaborativa. En efecto, este mecanismo permite enriquecer la visión particular de cada miembro del grupo, y de ese modo contribuir a una colaboración real.

Una adecuada combinación de ambas perspectivas de *consciencia de grupo* fomenta la colaboración en la línea de obtener un compromiso entre el grado de *awareness* y el uso de la red, identificado por Correa y Marsic en [CM03], tal y como se presenta en el *Capítulo 2* (véase *Capítulo 2*; *sección 2.2.5.*).

Una primera aproximación a este enfoque para proporcionar componentes de apoyo al trabajo en grupo desde la perspectiva de la plataforma cliente se encuentra en [SC06].

6.4. Otras iniciativas en el campo del contexto bajo el enfoque de la POA

No es extraño que existan otras iniciativas en el campo de la *sensibilidad al contexto* bajo el enfoque de la POA, dados sus numerosos beneficios. Esta sección describe estas iniciativas, enmarcándolas en la categoría de herramienta del contexto que le corresponde a cada una de ellas, de acuerdo a la clasificación realizada en el *Capítulo 5* (véase *Capítulo 5*; *sección 5.2.*).

De hecho, la idoneidad de un enfoque de *separación de conceptos* avanzado (e.g. POA, SOP o *filtros de composición*) para la identificación, especificación e implementación de competencias relacionadas con la personalización ya se pone de manifiesto en el trabajo de Mesquita et al. [MBdL02].

Según estos autores, los mecanismos de personalización (que incluyen tareas como el registro de un historial y técnicas de monitorización y seguimiento, entre otras) aparecen típicamente diseminados en el diseño de software. Defienden que una clara identificación de las competencias relacionadas con la personalización hace posible la incorporación a posteriori de los mecanismos necesarios en aplicaciones ya existentes. No obstante, plantean también como cuestión abierta el impacto que puede tener sobre la flexibilidad y mantenibilidad del diseño software la consideración de estos aspectos en las etapas tempranas de desarrollo.

Es en esta misma línea de estudio que se desarrolla el trabajo del grupo de investigación LIFIA (Laboratorio de Investigación y Formación en Informática Avanzada), perteneciente a la Universidad Nacional de La Plata, y también el que subyace en el campo conocido como *línea de productos*, presentados a continuación.

Por otro lado, existe un grupo en la Universidad Federal de Pernambuco de Brasil que ha definido un patrón arquitectural como soporte al desarrollo de aplicaciones adaptativas.

Otro enfoque muy distinto, es el que se está llevando a cabo por el grupo GISUM (Grupo de Ingeniería del Software de la Universidad de Málaga). Este grupo utiliza un enfoque que combina dos paradigmas de programación: el desarrollo de software basado en componentes (del inglés *Component-Based Software Development*; CBSD en adelante) [BW98] y el Desarrollo de Software Orientado a Aspectos (AOSD) para hacer frente a la complejidad de los sistemas distribuidos. Su esfuerzo de investigación ha estado enfocado hacia la definición y desarrollo de distintos marcos de trabajo para la ejecución de sistemas distribuidos sobre una plataforma de componentes y *aspectos* [PFT02]. Entre las distintas aplicaciones de su trabajo se encuentra el desarrollo de entornos virtuales colaborativos y la construcción de un middleware de soporte al desarrollo de aplicaciones en el campo de la inteligencia ambiental (AmI del inglés *Ambient Intelligence* en adelante).

6.4.1. Línea de productos

Una línea de productos implica ofrecer un grupo de productos estrechamente relacionados entre sí, pero que se ofrecen a la venta de forma individual [CN01].

6.4.1.1. ¿Qué es una línea de productos software?

Se define una línea de productos como un conjunto de productos que comparte una serie de requisitos, aunque también exhiben una variabilidad significativa en los mismos [Kru01]. Debe estar respaldada por una técnica de la ingeniería del software que soporta el desarrollo de una familia de sistemas, perteneciente a un sector particular del mercado, los cuales comparten una serie de características. Estos sistemas son desarrollados a partir de un conjunto común de componentes núcleo, los cuales pueden ser configurados de distintas formas siguiendo un método preestablecido. Este enfoque resulta apropiado para dirigir el problema de la gran variabilidad de dispositivos móviles y las numerosas diferencias técnicas que presentan, en un intento por conseguir programar una aplicación una sola vez y ejecutarla en dispositivos diversos sin requerir modificación alguna [AMC+07].

Uno de los sectores en los que la línea de productos está teniendo una especial aceptación y difusión es en el desarrollo de juegos para móviles, dada su significativa variabilidad en capacidad y funcionalidad, así como en la cantidad de APIs propietarias para las distintas marcas de dispositivo (las cuales exploran las características más avanzadas de los mismos como la manipulación de sonido y los gráficos avanzados), dando lugar a la problemática conocida como *fragmentación de API* (véase *Capítulo 2*; *sección 2.1.3.*) [AMC+05].

Un simple juego puede presentar numerosas variaciones funcionales, pudiendo tener que ser desplegado en un amplio conjunto de dispositivos. En consecuencia, el software resultante es altamente variable. Este tipo de aplicaciones debe afrontar toda esa variabilidad, que en ocasiones implica restringir las características de la aplicación a los recursos disponibles en cada dispositivo particular, teniendo que recurrir en ocasiones a deshabilitar ciertas funcionalidades. El manejo de esas variaciones juega un papel clave en el proceso de desarrollo, en el que un enfoque basado en la línea de productos resulta apropiado.

6.4.1.2. Enfoques existentes

Existen varios enfoques para el desarrollo de líneas de productos de software: proactivo, reactivo y extractivo [Kru01]. En el enfoque proactivo se trata de analizar, diseñar e implementar una línea de productos para soportar el ámbito completo de productos previsibles en un futuro. En cambio, en un enfoque reactivo, se opta por ir construyendo incrementalmente una línea de productos a medida que van surgiendo demandas de nuevos productos o de nuevos requisitos en los productos ya existentes. En el enfoque extractivo se trata de obtener y evolucionar una línea de productos simple a partir de productos ya existentes. Por lo tanto, estos dos últimos enfoques son inherentemente incrementales

[AMC+05]. Debido a que el enfoque proactivo exige una alta inversión y corre más riesgos muchas empresas optan por los otros dos enfoques.

En particular, dado que los requisitos de portabilidad que deben cumplir los juegos para móvil son considerablemente transversales (atraviesan y enmarañan el código, esto es, constituyen *conceptos transversales* –concepto definido en el *Capítulo 1* (véase *Capítulo 1*; *sección 1.3.*), recientemente han emergido varios enfoques y grupos de trabajo que proponen el desarrollo de líneas de productos siguiendo la tecnología de la POA, constituyendo hoy en día una de las técnicas más destacadas en este campo [AMC+07].

6.4.1.3. Líneas de productos basadas en POA

Se puede mencionar el trabajo desarrollado en la Universidad Federal de Pernambuco (Brasil), que presenta un método que combina los enfoques reactivo y extractivo a partir de diseños e implementaciones ya existentes [AMC+05]. Se trata de un enfoque incremental para refactorizar juegos en J2ME existentes, aislando las partes comunes de las variables utilizando *AspectJ*. Identifican los puntos de variación y posteriormente refactorizan ese código para ser encapsulado y extraído en *aspectos*. El resultado es un *aspecto* que introduce las partes específicas de cada plataforma, mientras que las partes comunes se concentran en una versión abstracta del juego de partida. Así, para cada juego desarrollado independientemente en distintas plataformas se llega a una línea de productos que contiene el juego abstracto y tantos *aspectos* como partes variables hayan sido identificadas. Esto se repetiría para cada nueva plataforma. Finalmente, una vez finalizado el proceso de factorización, cada conjunto de *aspectos* apropiado a cada plataforma debe componerse con el núcleo de la aplicación para obtener una versión desplegable.

El trabajo de Young [You05] extiende el trabajo anterior desarrollando una línea de productos para aplicaciones móviles que utiliza técnicas de orientación a *aspectos* empezando desde el principio, como enfoque opuesto a la refactorización de las aplicaciones existentes, siguiendo un enfoque proactivo. Su esfuerzo se centra por tanto en el desarrollo de aplicaciones completas considerando las capacidades de los dispositivos y encapsulando las distintas funcionalidades de los mismos en *aspectos*. De ese modo se pueden obtener varias versiones de la misma aplicación, pudiéndose reutilizar los *aspectos* en nuevas aplicaciones.

Por otro lado, este trabajo abarca un rango más amplio no sólo de tipo de aplicaciones –no se limitan al estudio de juegos para móviles-, sino también de dispositivos, incluyendo PDAs y *pagers*, con respecto a los trabajos previos.

En [You05] además se realiza un estudio comparativo tanto del código fuente como del proceso de construcción entre el enfoque propuesto y otra implementación estrictamente *orientada a objetos*, enfocando la comparativa específicamente en el problema de la *fragmentación de APIs*. Como resultado del análisis, concluyen que un enfoque proactivo basado en POA permite al desarrollador aislar eficientemente las APIs únicamente soportadas por un tipo específico de dispositivos, manteniéndolas en un módulo simple en lugar de estar dispersas a lo largo de la aplicación. Por lo tanto resulta ser un método adecuado para variar la funcionalidad disponible de una aplicación dependiendo del dispositivo destino.

En particular, nuestra propuesta se corresponde con un enfoque proactivo, tratando de diseñar e implementar por adelantado las posibles necesidades futuras, de manera predecible, ofreciendo ese código en forma de framework (véase *Capítulo 8*), opuesto a la refactorización de aplicaciones existentes. No obstante, el foco de atención no está fijado en la variabilidad en capacidad y funcionalidad de los dispositivos móviles, sino en una vertiente dinámica y más extensa de la variabilidad: la *sensibilidad al contexto*. En cierto modo se puede asimilar como una línea de productos para la construcción de *aplicaciones sensibles al contexto* para un conjunto prefijado de restricciones contextuales.

6.4.1.4. Categorización

En este tipo de productos, la decisión acerca de qué módulos componer para configurar una versión personalizada a un dispositivo concreto se resuelve en tiempo de compilación, constituyendo una decisión perteneciente a la fase de diseño. La aplicación resultante no proporciona ningún mecanismo de adaptación dinámica, no pudiendo por tanto ser considerada como una *aplicación sensible al contexto* (esto es, un sistema con *plasticidad implícita*). Se trata por tanto de un soporte para desarrollar aplicaciones *adaptables* a unas funcionalidades y potencialidades concretas, pero no *adaptativas*. En definitiva, el desarrollo de líneas de productos se enmarca como un soporte de *plasticidad explícita* (véase *Definición 2.5.*; *Capítulo 2 - sección 2.2.2*).

Por otro lado, el dominio de aplicación es muy específico abarcando tan sólo el ámbito de las aplicaciones de juegos para móviles en algunos casos, o bien aplicaciones para móviles en general, aunque restringido siempre al campo de la computación móvil. Cabe remarcar también que el objetivo de esta línea de trabajo se centra únicamente en la *adaptabilidad* a los recursos hardware relacionados con sus características estáticas (véase la clasificación de las variaciones en recursos hardware presentada en el *Capítulo 2*; *sección 2.1.3.*), no contemplando ningún factor más relativo al contexto, ni siquiera un perfil de usuario preestablecido.

6.4.2. Adaptación dinámica en AspectJ

El grupo LIFIA de la Universidad Nacional de La Plata dedica parte de sus esfuerzos a investigar el uso de *AspectJ* en el desarrollo de aplicaciones *adaptativas*, esto es, aplicaciones que modifican su comportamiento a lo largo de la ejecución de acuerdo a los cambios en su entorno. Su trabajo se enmarca en el desarrollo de *aplicaciones sensibles al contexto* (véase *Capítulo 5*; *sección 5.2.1.*), haciendo especial incidencia en la sensibilidad a los recursos hardware (ancho de banda, disponibilidad del servidor, nivel de batería, etc.).

6.4.2.1. Comparación con el trabajo propuesto en esta tesis

Proponen una misma estructura software dividida en tres capas: la aplicación base -donde no aparece ninguna referencia a las restricciones del entorno-, una *capa aspectual* que contiene el comportamiento adaptativo utilizando también la POA, y una tercera capa que alimenta a ésta con información dinámica. No obstante, el enfoque acerca de cómo lograr esa misma meta es muy distinto.

La estructura en capas de software propuesta está enfocada al desarrollo de *aplicaciones sensibles al contexto*, esto es, aplicaciones móviles concebidas desde un principio para ofrecer esta funcionalidad extra. Por lo tanto, desde las fases iniciales las capas adicionales (*aspectual* y *sensible al contexto*) van siendo programadas explícitamente para la capa del sistema, obteniendo como resultado un fuerte acoplamiento de aquéllas con respecto a la aplicación base.

En efecto, a pesar de que la capa base no se ve afectada por las capas adicionales, y por lo tanto, puede evolucionar en independencia de la *capa aspectual*, no ocurre lo mismo con la *capa aspectual*, que al trabajar con un modelo de *puntos de unión* basado en la signatura de los métodos se convierte en un estructura ligada al código base. Adaptar las capas adicionales a una aplicación distinta supone, en el mejor de los casos, la personalización de los *puntos de corte* a los métodos que forman parte de la aplicación base, lo que provoca un impacto a lo largo de toda la *capa aspectual*, al tener que ser adaptado *aspecto* por *aspecto* a cada nuevo sistema. En otros casos debe comenzarse de nuevo desde la fase de diseño.

En definitiva, bajo la opinión del autor de esta tesis, más que un artefacto software, lo que se ofrece es una estructura software y una metodología de trabajo para estructurar el diseño e implementación de las *aplicaciones sensibles al contexto* específicamente concebidas para unas restricciones contextuales determinadas.

Es aquí donde radica la diferencia con el trabajo propuesto en esta tesis, en el que se ofrece un framework genérico fácilmente instanciable en las capas adicionales (*aspectual* y *sensible al contexto*) que, una vez acopladas a una cierta aplicación ya existente, en la que no se maneja ningún tipo de restricción relativa al contexto, la dotan de la funcionalidad *sensible al contexto* especialmente seleccionada para la misma. El acoplamiento al que se está haciendo referencia cumple con la característica de ser (a) no invasivo, esto es, que no genera ningún impacto en la aplicación, preservando de ese modo la posible evolución de sus necesidades contextuales; y (b) localizado en un punto concreto de la *capa aspectual*, concretamente en el denominado *aspecto anotador* (se presenta en detalle en el *Capítulo 8* -véase *Capítulo 8*; *sección 8.2.1.1.*), minimizando por tanto también el impacto en las capas adicionales, las cuales mantienen un alto nivel de genericidad.

Una extensa explicación del framework (el *Framework de Plasticidad Implícita*) al que se está haciendo referencia aquí, basada en la arquitectura software presentada en este capítulo se encuentra en el *Capítulo 8*.

6.4.2.1.1. Otras diferencias

A continuación se relacionan otras observaciones extraídas del estudio de este trabajo las cuales, como se puede comprobar en el *Capítulo 8* destinado a la presentación del framework, constituyen aspectos diferenciadores.

- Por lo que respecta a la *capa sensible al contexto*, ésta se construye de manera totalmente ad hoc, esto es, agrupando los factores contextuales sin respetar el principio de ortogonalidad que sustenta la *capa aspectual*. En consecuencia, la evolución de la *capa sensible al contexto* a distintas necesidades contextuales resulta complicada.

  En el *Framework de Plasticidad Implícita* se adoptan esquemas de diseño para cada tipo de necesidad contextual, los cuales son reutilizables para cualquier aplicación, no siendo necesario, por tanto, empezar de cero en cada ocasión.

- La mayor parte del código encargado de la adaptación forma parte de la *capa aspectual*. Esto acarrea una serie de problemas:

  1. el desarrollador está forzado a conocer el paradigma de POA, y concretamente el lenguaje *AspectJ* para desarrollar *aplicaciones sensibles al contexto*.
  2. todo el código encargado de una restricción relacionada con el contexto queda encapsulado en una única entidad del programa –en este caso la entidad *aspecto*-, dificultando su legibilidad, así como también su evolución, por ejemplo, mediante la aplicación de patrones y otros principios de diseño del software.

Se puede decir, por tanto, que las buenas prácticas de diseño no se aplican en todas sus consecuencias.

3. por los motivos mencionados anteriormente, cuanto más código forme parte de la *capa aspectual* mayor es el acoplamiento con la aplicación base.
4. en realidad no existe una separación entre la adquisición del contexto y el uso que se hace del mismo, ambos concentrados en el código de los *aspectos*. En definitiva, la *separación de conceptos* en lo que respecta al tratamiento de las restricciones contextuales es mínima (véanse los requisitos de las herramientas *sensibles al contexto* recogidas en el *Capítulo 5*; *sección 5.2.2.*), lo que dificulta la reutilización del código *aspectual* a distintos tipos de contextos o sensores a utilizar.

En el *Framework de Plasticidad Implícita* el código encapsulado en los *aspectos* es muy conciso, destinándose a la invocación de la funcionalidad desarrollada en la *capa sensible al contexto*, que es la que realmente contiene las clases encargadas de las operaciones principales relacionadas con la adaptación contextual. Esta estrategia permite que esas clases pueden ser reutilizadas, a la vez que el código de los *aspectos* se simplifica, facilitando así su legibilidad y el mantenimiento de ambas partes.

- Tampoco se menciona en este trabajo el cumplimiento de otros requisitos propios de una *aplicación sensible al contexto*, tales como el del almacenamiento del contexto -véase *Capítulo 5*; *sección 5.2.2*, que sí se contemplan en el *Framework de Plasticidad Implícita*.

- Se centran únicamente en el desarrollo de aplicaciones móviles, abarcando exclusivamente el tratamiento de restricciones relacionadas con la movilidad y las restricciones de los dispositivos compactos. En el *Framework de Plasticidad Implícita* se tratan otro tipo de restricciones no necesariamente ligadas con aplicaciones móviles, proporcionando componentes específicas tanto para aplicaciones móviles como para aplicaciones convencionales.

Como ya se ha mencionado, el tipo de contexto manejado se centra principalmente en las restricciones en recursos hardware susceptibles de variación dinámica [ZVG06]. También contemplan el aspecto de la localización, combinado con aquéllas, [ZGJ04], y por último aspectos relativos al usuario [ZPG04]. Los atributos del contexto manejados en el *Framework de Plasticidad Implícita* son más amplios, tal y como se presenta en el *Capítulo 2* (véase *Capítulo 2*; *sección 2.1.3.*).

6.4.2.1.2. Aspectos relativos a la personalización

Para dar solución al aspecto de la personalización plantean una estructura software muy compleja relacionada más bien con una adaptación de contenidos que con una adaptación de la IU, con unas metas muy ambiciosas tanto en los objetivos a lograr como en el método a seguir. Lo previsible es que este módulo requiera ser construido en torno a una base de conocimientos y consuma gran cantidad de recursos.

Bajo la opinión del autor de esta tesis este planteamiento no es demasiado realista con las restricciones de los dispositivos móviles. Este trabajo se enmarca bajo el tópico de modelado de perfiles, llevado a cabo en trabajos como el de [KPRS03]. En definitiva, el tipo de personalización planteada por el trabajo del grupo LIFIA se corresponde más bien con el propósito de la *plasticidad explícita* (véase *Definición 2.5*; *Capítulo2 – sección 2.2.2.*) que con el de la *plasticidad implícita*, centrado en adaptaciones dinámicas y objeto de estudio en este capítulo.

Por otro lado, no queda constancia de qué tipo de heurísticas o técnicas emplean en el proceso de definición de la adaptación. Desconocemos que el trabajo del grupo LIFIA haya avanzado el aspecto relativo a la personalización, ni que dispongan de una versión implementada o prototipo que avale su propuesta de diseño.

Bajo la opinión del autor de esta tesis, el aspecto de personalizar una aplicación móvil debe ser muy conciso, enfocado hacia objetivos muy concretos y puntuales, o bien orientados hacia la IU, como pudiera ser el de personalizar los valores por defecto de un formulario o bien ordenar una lista en base a las preferencias o selecciones realizadas previamente por el usuario a la hora de interaccionar con ella, o bien orientado hacia el ajuste de ciertos parámetros asociados a funcionalidades concretas. Estos son precisamente los objetivos de personalización trabajados en los casos de estudio llevados a cabo (véase *Capítulo 7*).

6.4.2.2. Categorización

Al no proporcionar un soporte generalizado para satisfacer las necesidades y la problemática general de la *computación sensible al contexto*, este trabajo se enmarca en la categoría de *aplicaciones sensibles al contexto.*

La falta de generalidad hace que resulte casi imposible integrar soluciones ya existentes a aplicaciones que no son *sensibles al contexto*, o incorporar nuevo contexto a las que ya lo son.

El tipo de contexto es, como se ha mencionado, el relacionado con recursos hardware mayoritariamente, y también la localización y personalización al usuario.

6.4.3. Patrones arquitecturales para estructurar aplicaciones adaptativas

Uno de los trabajos más próximos al presentado en la presente tesis –en concreto con el *Framework de Plasticidad Implícita* presentado en el *Capítulo 8*- es el desarrollado por un subgrupo perteneciente también a la Universidad Federal de Pernambuco de Brasil. Su esfuerzo se dirige hacia la minimización de los problemas que acarrea la aplicación de las denominadas arquitecturas *Adaptive Object-Model* (AOM) -un tipo particular de arquitectura reflexiva que representa entidades como *metadatos*, definidas en el *Capítulo 5*; *sección 5.3.2.*- [YJ02]. En concreto, proponen el uso de POA para conseguir que los sistemas Adaptive Object-Model sean más fáciles de evolucionar y mantener, especialmente con respecto a la inclusión de nuevos requisitos adaptativos y, en definitiva, haciendo este tipo de sistemas más adaptables y extensibles. Esta problemática ha sido enunciada de la forma siguiente:

> *"a pesar de que los sistemas Adaptive Object-Model son adaptativos, no son, sin embargo adaptables" [Lie95].*

Definen un patrón arquitectural denominado *Adaptability Aspects*, donde a través de los *aspectos* se define cómo cambiar el comportamiento de las funcionalidades de la aplicación para soportar adaptatividad.

Las ventajas de utilizar POA en sistemas Adaptive Object-Model pueden resumirse en las siguientes: (1) facilita posibles cambios en los puntos de la ejecución donde deben obtenerse datos dinámicos; y (2) aísla las acciones de adaptabilidad de la lógica de negocio y del código de la interfaz [DYBJ04], aportando mayor modularidad.

6.4.3.1. Descripción del patrón arquitectural 'Adaptability Aspects'

El objetivo del patrón arquitectural *Adaptability Aspects* es el de mejorar la modularidad de los sistemas Adaptive Object-Model a la hora de estructurar aplicaciones adaptativas, utilizando POA, como una de las tecnologías más destacadas de separación de conceptos. Para ello se encarga de controlar y modificar el comportamiento de la aplicación base, especificando cómo cambiar la funcionalidad para adaptarla a los cambios contextuales.

Identifica cinco componentes básicas: (1) *la aplicación base*, que contiene la funcionalidad principal de la aplicación; (2) *clases auxiliares*, en las que se delegan las distintas tareas relacionadas con la adaptación; (3) *aspectos de adaptabilidad*, los cuales invocan los métodos de las clases auxiliares; (4) el *gestor del contexto*, responsable de monitorizar

y analizar cambios contextuales, permitiendo que puedan ser soportados nuevos mecanismos para acceder al contexto sin provocar un impacto significativo en la aplicación; y (5) *módulo proveedor de datos de adaptación*, el cual contiene las clases responsables de proporcionar los datos dinámicos que especifican cómo adaptar la aplicación ante una determinada situación. Un mismo cambio contextual puede dar lugar a distintos comportamientos en distintos momentos de acuerdo a los datos proporcionados por este módulo. Estas clases son organizadas como un Adaptive Object-Model. Son las clases auxiliares las que se comunican con este módulo.

La figura 6.4 muestra una representación de los módulos integrantes y de su interrelación utilizando la notación de paquetes UML, donde cada paquete representa cada una de los componentes lógicos descritos.

Este patrón ha sido aplicado en dos experimentos llevados a cabo con J2ME: un diccionario y un álbum de fotografías, ambos desarrollados para dispositivos móviles.

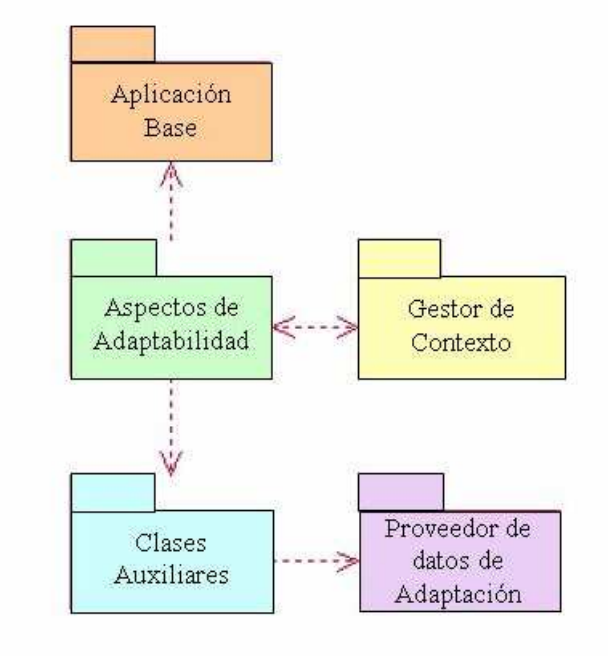


Figura 6.4: Componentes del patrón arquitectural *Adaptability Aspects*.

6.4.3.2. Comparación con el trabajo propuesto en esta tesis

El objetivo de este artefacto software es el mismo que el propuesto en esta tesis para el *Framework de Plasticidad Implícita*: facilitar la construcción de *sistemas adaptativos*

obteniendo una separación de conceptos lo más nítida posible, tratando de optimizar la extensibilidad, reutilización y facilidad de mantenimiento del sistema resultante.

Una de sus características destacadas es que delega la mayor parte del código encargado de la adaptación a clases implementadas en POO, las denominadas clases *auxiliares* en este patrón, y que se corresponden con las clases de la *capa sensible al contexto* de la arquitectura software propuesta en este capítulo, con las consecuentes ventajas (básicamente reutilización y el no requerir conocer la tecnología de la POA por parte del desarrollador), mencionadas anteriormente. Como se ve en el capítulo de presentación del *Framework de Plasticidad Implícita –Capítulo 8-*, también se destina la mayor parte del código de adaptación a la *capa sensible al contexto*, que es la que contiene realmente el entramado de clases encargadas de manejar las adaptaciones.

Otra similitud es la estructura de las distintas componentes software que intervienen. En especial, la arquitectura propuesta en esta tesis se corresponde con la estructura correspondiente al eje central de la figura 6.4 (*aplicación base*, *aspectos de adaptabilidad* y *clases auxiliares*).

6.4.3.2.1. Comparación relativa a las consecuencias de la aplicación del patrón

A continuación se exponen las consecuencias positivas y negativas en el diseño derivadas de la aplicación del patrón, indicando si el *Framework de Plasticidad Implícita* comparte o no cada una de ellas.

Beneficios

Este patrón proporciona los siguientes beneficios: modularidad, reutilización, extensibilidad, cambios dinámicos e independencia de plataforma, aunque en este último caso cabe tener en cuenta ciertas restricciones en cuanto a tamaño de código y eficiencia de cara a ser implementado para un dispositivo compacto, así como ciertas restricciones de capacidad de carga dinámica que no todos los lenguajes soportan.

El *Framework de Plasticidad Implícita* propuesto en la presente tesis comparte la mayor parte de los beneficios mencionados. No hay que descuidar que los aspectos de modularidad, reutilización, ortogonalidad y transparencia son las metas de partida que han estado presentes desde su concepción. El aspecto de independencia de plataforma también constituye una de las preocupaciones principales en el desarrollo del *Framework de Plasticidad Implícita*. De hecho, en aquellos componentes en los que no es posible evitar ciertas dependencias con el perfil para dispositivos móviles MIDP[8] (Mobile Information Device Profile),

[8]Especificación para el uso de Java en dispositivos compactos tales como teléfonos móviles y PDAs. Forma parte de la implementación J2ME.

o simplemente resulta más beneficioso en cuanto a reutilización de código no generalizar tanto, el *Framework de Plasticidad Implícita* ofrece dos versiones para la componente en cuestión: la versión para aplicaciones móviles y la versión para aplicaciones fijas.

Cabe remarcar que en relación a los cambios dinámicos la concepción es muy distinta. Así como en el patrón *Adaptability Aspects* se plantea la necesidad de ir cargando nuevos mecanismos de adaptación conforme se van encontrando distintas situaciones contextuales, el tipo de adaptaciones que se plantean para el *Framework de Plasticidad Implícita* son bajo un punto de vista de *close-adaptiveness* [OMT98], esto es, auto-contenido -se incluyen todos los mecanismos necesarios para llevar a cabo la adaptación de manera autónoma- (véase *Capítulo 2*; *sección 2.2.3.*). Por lo tanto no se plantea la necesidad de soportar la adición de nuevos comportamiento durante la ejecución. De eso se encarga el *Motor de Plasticidad Explícita* (véase *Capítulos 3* y *4*).

Precisamente esta es la idea que subyace en el modelo de la *visión dicotómica de plasticidad* propuesta en esta tesis, y que precisamente distingue entre adaptaciones autónomas -*plasticidad implícita*- y adaptaciones que no pueden ser soportadas por el propio dispositivo, y por tanto son resueltas en el servidor –*plasticidad explícita*. En lugar de recurrir a la carga dinámica de nuevos módulos encargados de un nuevo tipo de adaptación se recurre al servidor, que ofrece la potencia necesaria para afrontar adaptaciones de gran complejidad impensables de resolver en el propio dispositivo sin provocar interrupciones en la ejecución. En consecuencia, el comportamiento adaptativo a abordar por el *Framework de Plasticidad Implícita* debe ser programado antes de su despliegue.

En efecto, la misión del *Motor de Plasticidad Implícita*, tal y como se concibe bajo la *visión dicotómica*, es la de resolver aquellas adaptaciones preestablecidas de antemano, para unas condiciones y unos atributos del contexto concretos. Ciertamente, es ante circunstancias no previstas que se recurre al servidor, donde se decide si conviene cargar nuevos módulos más especializados para las nuevas circunstancias, aunque ello suponga la interrupción de la ejecución en determinados casos, tal y como se explica en el *Capítulo 8* (véase *Capítulo 8*; *sección 8.4.2.*).

Responsabilidades

A pesar de las considerables ventajas que aporta este patrón, su aplicación comporta también ciertos riesgos, sobretodo teniendo en cuenta que las aplicaciones para dispositivos móviles son las que acaparan mayor atención en el campo de las *aplicaciones sensibles al contexto*. A continuación se presentan los compromisos que presenta este patrón.

- *Tamaño del código.* La implementación de este patrón requiere más código que otras soluciones alternativas no modulares. En muchas ocasiones esto puede representar un problema para los sistemas embebidos, en los que los recursos son limitados.

  Utilizando el framework propuesto en esta tesis el tamaño del código resultante también se ve incrementado, aspecto que es analizado experimentalmente en el *Capítulo 7*. No obstante, el número de componentes, envergadura de las mismas y, en definitiva, tamaño de código final, es considerablemente inferior al que resulta de la implementación del patrón *Adaptability Aspects*.

- *Eficiencia.* Se trata de una implementación compleja que utiliza un elevado número de componentes, pudiendo provocar el rechazo por parte del desarrollador, más propenso a adoptar soluciones más simples, a pesar de no ser modulares. Esta cuestión es aún más problemática en el caso de aplicaciones para dispositivos compactos.

  Por otro lado, el hecho de que las clases del *módulo proveedor de datos de adaptación* estén organizadas como un Adaptive Object-Model contribuye también a reducir la eficiencia puesto que, en definitiva, se trata de una arquitectura meta-nivel, la cual debe resolver adicionalmente la interpretación de metadatos. Sólo está justificado recurrir al uso de AOM cuando el grado de adaptabilidad requerido es importante. Ya se ha discutido arriba que este grado de adaptabilidad no es tan prioritario en un enfoque basado en la *visión dicotómica*, y por lo tanto en un *Framework de Plasticidad Implícita* concebido para operar bajo este modelo.

  Tal y como ya se ha comentado, el número de componentes a integrar utilizando el *Framework de Plasticidad Implícita* es inferior al propuesto en este patrón arquitectural, lo que se traduce en una menor complejidad. Por otro lado, la misión de un framework es la de proporcionar gran parte del código listo para ser utilizado, evitando por tanto la codificación de la mayor parte de los componentes, o incluso de la mayor parte del código de los componentes incompletos, tal y como se detalla en el *Capítulo 8*.

- *Carga dinámica.* En este patrón el nuevo comportamiento de la aplicación se presenta a través de metadatos, en lugar de hacerlo fijando las adaptaciones en el código. Sin embargo, en ocasiones no es factible la carga de código dinámicamente con objeto de incluir nuevos mecanismos de adaptación sin interrumpir la aplicación. Esto depende de la plataforma de la aplicación, así como del lenguaje de POA utilizado. Así, por ejemplo, *AspectJ* no permite la carga dinámica de nuevos *aspectos*. Tampoco J2ME está provista de las clases que proporcionan adaptabilidad dinámica, tal como *ClassLoader*, ni tampoco del paquete de reflexión.

Cabe destacar que la aplicación del *Framework de Plasticidad Implícita* no requiere hacer uso de la carga dinámica, consiguiendo igualmente que el comportamiento adaptativo de la aplicación no aparezca fijado en el código núcleo de la aplicación.

6.4.3.2.2. Diferencias en cuanto a su concepción

En la sección previa se han tratado una a una cada una de las diferencias funcionales entre el patrón arquitectural *Adaptability Aspects* y el *Framework de Plasticidad Implícita* presentado en esta tesis, así como las derivadas de su aplicación. No obstante, la mayor diferenciación entre ambos trabajos responde a una diferencia conceptual, la cual determina todo su desarrollo y su consecuente comprensión. En efecto, se está contrastando la aplicación de un *patrón arquitectural* con respecto a la aplicación de un *framework*. Argumentar acerca de esta distinción constituye una discusión no trivial basada en la concepción de ambos tipos de artefactos software, muy extendidos en el desarrollo de software actual.

Por un lado, se define un *patrón de diseño* como una solución a un problema de diseño no trivial, la cual es efectiva (ya se resolvió el problema satisfactoriamente en ocasiones anteriores) y reutilizable (se puede aplicar a diferentes problemas de diseño en distintas circunstancias) [GHJV95]. En particular, un *patrón arquitectural* expresa un esquema organizativo fundamental para estructurar los sistemas software [BMR+96].

Básicamente un patrón de diseño es una idea abstracta, una orientación respaldada y probada en la construcción de módulos software. No obstante, la reutilización que se ofrece es en cuanto a la idea y estructura de la solución. No se ofrece código directamente reutilizable en un sistema concreto. Se han erigido como una manera de compartir conocimiento y experiencia en el desarrollo de soluciones software, estableciendo un vocabulario común y un estándar en la manera como transmitir ese conocimiento.

Un *framework*, sin embargo, se puede describir como una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Es el esqueleto sobre el cual varios objetos son integrados para ofrecer una determinada solución [FSJ99]. Otra definición de framework es la que lo considera como un conjunto de componentes integradas que colaboran para proporcionar una arquitectura reutilizable para una familia de aplicaciones relacionadas [Sch06].

Típicamente, un framework puede incluir soporte de programas, bibliotecas y en ocasiones un lenguaje de *scripting* con objeto de ayudar a desarrollar y unir los diferentes componentes de un proyecto. Pueden consistir en un conjunto de herramientas, utilidades,

interfaces o clases que añaden una capa de abstracción al sistema subyacente sobre el que va a ser instanciado. Los frameworks son diseñados con la intención de facilitar el desarrollo de software, permitiendo a los diseñadores y programadores pasar más tiempo identificando requisitos de software que programando ciertas partes de código reutilizables. Ellos en sí mismos no son aplicaciones, sino que son construcciones más complejas.

La principal diferencia, por lo tanto, entre *patrón de diseño* y *framework* es el hecho de que, tal y como se deduce de las definiciones previas, un patrón no constituye una solución en sí mismo. En cambio, un framework constituye una solución finita o parcial que ayuda a estructurar el código, ofreciendo un producto parcialmente desarrollado, válido para un determinado tipo de problemas. Alcanzar la versión definitiva y lista para su entrega es mucho más sencillo y directo mediante el uso de frameworks. Tan sólo es necesario completar ciertas partes de código y establecer los puntos de enganche necesarios para su instanciación en la aplicación.

Habitualmente un framework se compone de varios patrones de diseño utilizados para implementar ciertas partes del mismo. Los patrones no conducen a la reutilización directa de código, mientras que los frameworks sí. Esto acarrea una responsabilidad mucho mayor para los desarrolladores de frameworks, al tener que evitar las dependencias del sistema con objeto de conseguir su reutilización.

Según Markiewicz et al. en [MdL01], los frameworks están ganando rápidamente aceptación debido a su capacidad para promover la reutilización tanto del diseño como del código fuente. De acuerdo a estos autores, los frameworks se pueden concebir como generadores de aplicaciones que se relacionan directamente con un dominio específico, es decir, con una familia de problemas relacionados. En efecto, los frameworks son capaces de generar aplicaciones personalizando sus requisitos particulares.

6.4.4. Plataformas de soporte al contexto distribuido desarrolladas en el grupo GISUM

El propósito del trabajo desarrollado por el grupo GISUM es el de mejorar la modularidad proporcionada por un enfoque basado en componentes, a fin de alcanzar un mayor grado de reutilización, adaptabilidad y extensibilidad en el desarrollo de plataformas y middlewares de soporte a las aplicaciones distribuidas adaptativas [FJP05]. En particular, para la creación de middlewares ampliamente utilizables en sistemas embebidos distribuidos, es indispensable proporcionar un conjunto completo y fácilmente personalizable de servicios para afrontar las limitaciones de memoria de este tipo de sistemas, aspecto en el que la POA puede ofrecer una solución válida [PHK03].

Para el caso concreto de aplicaciones de AmI, la evolución dinámica se materializa en la gran variedad de dispositivos con distintas capacidades y recursos hardware. En el trabajo del grupo GISUM se propone el uso de AOSD para minimizar el impacto de la evolución, así como para abordar otros retos impuestos por el campo de la AmI, como son el de afrontar de manera transparente y dinámica la heterogeneidad en los datos, en el hardware y en la comunicación, así como la capacidad de soportar la adaptación de las aplicaciones AmI a diversos cambios dinámicos, entre los que se encuentra el descubrimiento de nuevos dispositivos [FJ05].

En este sentido han definido una plataforma denominada DAOPAmI [FJP05] que combina los beneficios mutuos de las tecnologías CBSD y AOSD al servicio de aplicaciones AmI, en un afán por resolver parte de las problemática de la programación de este tipo de entornos.

Otro ejemplo lo constituye la plataforma CoopTEL [AFJP04], que facilita la implementación de herramientas colaborativas a partir de la composición de componentes y *aspectos* previamente desarrollados.

Cabe decir que este trabajo aborda la evolución dinámica de los sistemas software desde el punto de vista de las características de distribución, y no especialmente en lo que concierne a los mecanismos de adaptación, como ocurre por ejemplo en el trabajo de Birov [Bir04], donde el dinamismo en la funcionalidad constituye una de las preocupaciones principales. Birov propone una arquitectura orientada a patrones para aplicaciones móviles adaptativas distribuidas donde se fomenta la modularidad de la aplicación a través de la separación de los requisitos de distribución y también de adaptación en unidades de programa *aspecto*.

6.4.4.1. Categorización

Por supuesto, este trabajo se enmarca dentro de la categoría de middlewares de soporte para el desarrollo de aplicaciones y entornos distribuidos. El tipo de problemática abordada es totalmente distinta. Se trata del desarrollo de plataformas a gran escala. Las metas abordadas no están centradas en la adaptación del comportamiento o interfaz de una aplicación concreta de acuerdo a ciertas circunstancias contextuales, sino en facilitar el desarrollo de la infraestructura subyacente a entornos distribuidos *sensibles al contexto*, abordando problemas como el de cambiar la configuración de una aplicación en tiempo de ejecución, como una de las posibles manifestaciones del problema de la evolución dinámica. La perspectiva abordada con respecto al problema de la *sensibilidad al contexto* en este caso es muy distinta.

6.5. Resumen y conclusiones del capítulo

El propósito de este capítulo es abonar el terreno para facilitar la comprensión y presentación en detalle del artefacto software desarrollado en esta tesis como soporte a la *plasticidad implícita*, tal y como se propone bajo la *visión dicotómica de plasticidad*. Este artefacto consiste en un framework de aplicación genérico, denominado *Framework de Plasticidad Implícita*. La descripción detallada de los componentes desarrollados se presenta en el *Capítulo 8*. En el *Apéndice B* se proporcionan los detalles de los esquemas de solución que han dado lugar a los componentes genéricos del framework. Por último, en el *Apéndice C* se recogen los diagramas de clases que representan cada una de los componentes del framework.

Se ha descrito con precisión el problema a cubrir, la aplicabilidad del framework, los objetivos y sus requisitos tanto funcionales como no funcionales. Estos últimos establecen las propiedades a alcanzar, cuyo planteamiento ha sido decisivo en las cuestiones de implementación. También se ha descrito la arquitectura software a la que da lugar la instanciación del framework en aplicaciones concretas, a fin de comprender el grado de cumplimiento de todas esas propiedades y metas marcadas ya desde su concepción. Por último, se han clarificado y definido los enfoques y técnicas aplicados.

La meta del *Framework de Plasticidad Implícita* es la de proporcionar funcionalidad *sensible al contexto* a aplicaciones ya desarrolladas, las cuales no fueron concebidas inicialmente para ofrecer esta característica, sin producir ningún tipo de alteración en la estructura y código de la aplicación objetivo. Este planteamiento descarta tanto enfoques de refactorización como enfoques pertenecientes a la disciplina de Desarrollo de Software Orientado a Aspectos, en los que se reconocen y consideran ya desde un principio las *competencias transversales*, las cuales son tratadas como *aspectos tempranos*. Bajo las premisas planteadas, la funcionalidad *sensible al contexto*, que efectivamente constituye una *competencia transversal*, tan sólo puede ser tomada en consideración a nivel de programación utilizando una técnica de programación basada en el enfoque de la *separación de conceptos*.

A pesar de que este enfoque amplía el campo de aplicación, siempre y cuando se consiga preservar la independencia con el sistema, tiene un precio que pagar: el tener que afrontar el acoplamiento de los componentes del framework, desarrollados de manera independiente, con la aplicación. No es un paso sencillo, puesto que requiere disponer de un conocimiento más o menos profundo de la aplicación. Sin embargo, la funcionalidad del contexto, que puede resumirse en la mecanización de tres pasos: detección, recogida y uso de la información contextual, involucra aspectos muy puntuales de la lógica de la

aplicación, lo que facilita la identificación de ciertos métodos como *puntos de enlace* en los que incidir para integrar los mecanismos de adaptación. En consecuencia, el nivel de conocimiento requerido no deja de ser relativamente superficial. El uso de *anotaciones de metadatos* para establecer los *puntos de enlace* de los *aspectos* con las clases es crucial para desacoplar estas componentes, un problema que no aparece resuelto en la literatura.

Por otro lado, de los tres pasos que caracterizan una funcionalidad del contexto, la detección y recogida de la información contextual puede ser implementada de forma totalmente independiente de la lógica de la aplicación cuando se trata de atributos del entorno o relacionados con los recursos hardware. Es sólo en el tratamiento de los aspectos relacionados con el usuario y con el grupo que se requiere un mayor conocimiento y acoplamiento con la lógica funcional, a efectos de monitorizar e inferir esa información. En el *Capítulo 8* se detalla la manera como se ha atacado esta problemática con el fin de preservar la genericidad y reutilización también en estas componentes del contexto. De cualquier modo, la implementación de todo el código de tratamiento queda delegada en una capa auxiliar (la *capa sensible al contexto*). En consecuencia, el contexto es modelado y, cuando así es posible, también capturado utilizando objetos a un nivel de abstracción apropiado, tal y como se realiza en [SDA99]. Los dos beneficios derivados de esta estrategia son: (1) incrementar la flexibilidad y reutilización de los componentes del contexto; y (2) facilitar la comprensión del framework por parte de los desarrolladores, liberándolos de tener que dominar la técnica de POA.

Este es uno de los puntos fuertes en la implementación del framework. En consecuencia, el objetivo de la *capa aspectual* está centrado en proporcionar un 'enganche transparente' entre el código base y las clases auxiliares, enlazando ambas funcionalidades, constituyendo por tanto el núcleo del *Motor de Plasticidad Implícita*. En realidad, este uso de los *aspectos* ha sido considerado ya como una buena práctica en [MWB+01].

En cuanto al análisis de las alternativas más conocidas en el ámbito de las técnicas de *Separación de Conceptos*, se deduce que la POA constituye el enfoque más adecuado para el tipo de problema a resolver, y también el más eficiente, abstracto y elegante, sin olvidar que también resulta considerablemente más fácil de entender que, por ejemplo, la *programación meta-nivel*. De hecho, así queda justificado tanto desde el punto de vista de la idoneidad con respecto al campo de aplicación, que incluso abre las puertas a los sistemas ubicuos, como desde el punto de vista de los beneficios en el diseño de software y la satisfacción en las propiedades exigidas desde un principio. Aunque la discusión en detalle de los resultados experimentales se presenta en el *Capítulo 7*, todas estas afirmaciones vienen a avalar la *Hipótesis 4* (Véase *Capítulo 1*; *sección 1.3.*).

En cuanto al esfuerzo requerido para dotar a las aplicaciones de funcionalidad *sensible al contexto*, cabe diferenciar los distintos grados de reutilización que aportan tres de los artefactos software más ampliamente extendidos: (1) arquitecturas software; (2) patrones de diseño; y (3) frameworks. En el caso de las arquitecturas lo que se ofrece es una orientación con respecto a la manera como estructurar los distintos módulos, así como la manera como establecer un mecanismo de interacción entre los mismos. Esta categoría se equipara con la de los patrones arquitecturales. En resumen, aunque el grado de orientación es mayor en el caso de patrones de diseño que en el caso de arquitecturas software, igualmente la implementación debe iniciarse de cero. Por lo que respecta a la distinción entre patrón y framework, mientras que un patrón es una solución lógica de diseño a un tipo de problema dado, un framework es una pieza de código física y, por tanto, directamente utilizable.

Los frameworks proporcionan código fuente listo para ser utilizado, al tiempo que identifican y localizan aquellas partes que requieren especialización. Esta distinción resulta verdaderamente abismal, sobretodo si el código reutilizable es el que predomina. Es obvio que el tiempo empleado para llegar al producto final se reduce espectacularmente. La curva de aprendizaje requerida para la comprensión del framework pronto se ve recompensada, más aún si el grado de aplicabilidad es elevado. Esta postura viene a contrarrestar una de las mayores críticas dirigidas a los sistemas *sensibles al contexto* en general, y en particular a los *sistemas adaptativos*. Ésta pone de relieve el dudoso equilibrio obtenido entre el esfuerzo empleado y la satisfacción obtenida. En ese sentido, el uso de frameworks está empezando a ganar aceptación debido a su capacidad para promover la reutilización tanto en el diseño como en el código fuente.

Por otro lado, cabe señalar que el *Framework de Plasticidad Implícita* está concebido como una herramienta (un toolkit) abierta, que gracias a la modularidad con la que ha sido concebida puede seguir creciendo, admitiendo una constante evolución. Eso significa que pueden incorporarse nuevos tipos de contexto, tantas necesidades contextuales y mecanismos de adaptación como se desee, mediante la organización y despliegue del framework como una librería jerárquica de *aspectos* y clases. A pesar de que hace falta comprobar empíricamente la satisfacción de los desarrolladores y el esfuerzo requerido para conocer, comprender y aplicar el framework en otras aplicaciones reales, esta particularidad hace pensar en la posibilidad de ser incluido en el campo del diseño de *aplicaciones sensibles al contexto*.

Por último, cabe destacar la novedad del framework presentado en esta tesis. En efecto, la revisión del estado del arte demuestra que, efectivamente, no se han propuesto otros frameworks de aplicación para la obtención de *aplicaciones sensibles al contexto* bajo las condiciones expuestas aquí y en el nivel de reutilización alcanzado.

Otros intentos por lograr reutilizar *aspectos* se presentan de manera informal como casos prácticos que, por otro lado, señalan como situaciones de reutilización problemática algunos puntos que, por primera vez, quedan resueltos utilizando el enfoque seguido en el *Framework de Plasticidad Implícita*. De nuevo, todas estas estrategias aparecen detalladas en el *Capítulo 8*. Otra de las limitaciones de los trabajos basados en POA analizados es que no se desarrollan formalismos que permitan discernir fácilmente en qué casos se puede reutilizar el comportamiento definido en los *aspectos* y a qué coste. A lo largo de los dos capítulos posteriores se pretenden concretar estos aspectos. En particular, el coste que supone aplicar el *Framework de Plasticidad Implícita*, en relación al beneficio de reutilización obtenido sí ha sido tratado en más o menos profundidad en este capítulo. No obstante, únicamente puede llegarse a valorar realmente una vez presentado el framework en detalle, y por supuesto, en un mayor grado de conocimiento, una vez utilizado para el desarrollo de aplicaciones concretas.

En general, las iniciativas en el campo de la *sensibilidad al contexto* haciendo uso de POA no proporcionan un soporte generalizado para satisfacer las necesidades ni la problemática general de la *computación sensible al contexto*, o bien responden a un planteamiento muy distinto. Quizás el trabajo más afín con el desarrollado en esta tesis es el del patrón arquitectural 'Adaptability Aspects'. En primer lugar, tal y como es concebido, el grado de reutilización de este enfoque es tan sólo a nivel de diseño. En segundo lugar, tal y como se plantea la *adaptatividad*, basada en el uso de un Adaptive Object-Model para proporcionar dinamismo, adolece de serias restricciones de eficiencia y de envergadura de código, las cuales plantean serios problemas en cuanto a su implantación en dispositivos compactos, así como ciertas restricciones de capacidad de carga dinámica que no todos los lenguajes soportan. En ese sentido, el enfoque propuesto en esta tesis, basado en la *visión dicotómica de plasticidad*, procura otorgar el grado de autonomía adecuado a cada plataforma, aunque haciendo prevalecer, si es necesario, la 'ubicuidad' por encima de aquélla. Si el tipo de contexto a manejar varía, existe la posibilidad de recurrir al s*ervidor de plasticidad*, que es quien reúne toda la biblioteca de componentes, tal y como se expone en el *Capítulo 8* (véase *Capítulo 8*; *sección 8.3.2.*).

La literatura concluye que para alcanzar el requisito de *sensibilidad al contexto* los diseñadores deben proporcionar arquitecturas flexibles que permitan adaptar el sistema a los requisitos cambiantes con facilidad. No obstante, en los enfoques de adaptación basados en arquitectura generalmente las distintas capas que componen el sistema son dependientes entre sí, de manera que los cambios en una de ellas habitualmente afectan a las otras. A pesar de que está universalmente reconocido que los enfoques basados en técnicas de reflexión permiten modificar el comportamiento y la estructura de una aplicación manteniéndola separada de los mecanismos de adaptación, el uso de técnicas de

POA optimiza la independencia entre ambos al modularizar mejor la funcionalidad que se entrecruza a lo largo de todo el código de la aplicación [CPA04].

Capítulo 7

Casos de Estudio y Experimentación

"Learn from yesterday, live for today, hope for tomorrow. The important thing is not to stop questioning."

Albert Einstein

Este capítulo está destinado a describir los distintos desarrollos llevados a cabo con el fin de probar la factibilidad de la arquitectura software propuesta para el *Motor de Plasticidad Implícita* en aplicaciones y dispositivos reales. En primer lugar se describen en detalle las aplicaciones de partida utilizadas, los objetivos perseguidos con los nuevos componentes del contexto, y el diseño de cada uno de los componentes desarrollados para satisfacer distintos atributos del contexto.

A continuación se describen las pautas seguidas, tanto a nivel general como a nivel específico en la construcción de cada componente, teniendo en cuenta las distintas necesidades identificadas en ambos casos de estudio, a fin de comenzar a perfilar esquemas de solución genéricos, como primer paso de abstracción para el diseño de un framework fácilmente personalizable.

Finalmente se presenta la parte experimental llevada a cabo. El propósito de la experimentación ha sido doble. En primer lugar verificar que la arquitectura propuesta es factible para aplicaciones destinadas a dispositivos compactos (tipo móviles 2G). En segundo lugar, contrastar esta implementación con la resultante de aplicar estrictamente técnicas de Programación Orientada a Objetos, a fin de estudiar no sólo el tamaño de la aplicación resultante, sino también los beneficios obtenidos en relación a la reutilización de código y efectos en la aplicación base, así como el consecuente impacto en la calidad software de las aplicaciones resultantes. Una discusión al respecto finaliza esta parte.

7.1. El Caso de Estudio *'Controlador de Obras'*

En una empresa del campo de la construcción ha sido implantada una aplicación móvil, cuyo objetivo es asistir a los trabajadores en su tarea diaria, así como servir de soporte para controlar el estado de su trabajo en curso.

7.1.1. Descripción del sistema de partida

A través de un teléfono móvil de uso personalizado, cada trabajador dispone diariamente de una descripción detallada de las tareas a desarrollar a lo largo de la jornada. Además, la aplicación le ofrece la posibilidad de introducir todo tipo de información relevante relacionada con su jornada laboral, como es el material empleado, la temporización de las tareas, la mano de obra utilizada, así como posibles incidencias que hayan podido surgir. Toda esa información se sincroniza con el sistema central al final de la jornada, con objeto de informar al departamento administrativo. Éste, en función de lo sucedido, reorganiza la labor a ser asignada a sus trabajadores para la jornada siguiente.

Esta es la funcionalidad núcleo del sistema de partida, el cual no incorpora capacidades de sensibilidad al contexto, ni tampoco está ideado para ofrecer ningún tipo de apoyo al trabajo en grupo, a pesar de estar concebido para operar en un entorno colaborativo. En efecto, el desarrollo de las obras por parte de la empresa consiste en una actividad colaborativa que requiere de coordinación, comunicación y colaboración por parte de todos sus empleados, y en la que se comparte una meta común: la completitud de todos los encargos lo más rápidamente posible. En lo sucesivo se hace referencia al sistema descrito como "ControladoraObras".

Es evidente que bajo un punto de vista y un *modus operandi* estrictamente individual no se obtiene toda la efectividad que se desearía para este tipo de sistema, propio de un entorno colaborativo, tal y como se pone de manifiesto a través de los tres escenarios de trabajo seleccionados y planteados más adelante. Generalmente hablando, el simple hecho de que cualquier tipo de incidencia no pueda ser transmitida hasta la finalización de la jornada, en el momento de realizar la sincronización, y siempre centralizando la información a través del departamento administrativo acarrea numerosas deficiencias de organización, dado que, impide la posibilidad de reorganizar tareas a lo largo de la jornada, conforme se van desencadenando los distintos eventos. Los trabajadores no tienen la opción de conocer el progreso del resto de sus colegas, ni la situación de la actividad colaborativa a lo largo de la jornada.

7.1.2. Descripción del sistema aumentado

A continuación se describe la funcionalidad extra introducida por el *Motor de Plasticidad Implícita* desarrollado para el sistema descrito. El sistema resultante aumenta la funcionalidad del sistema de partida, que de acuerdo a la estructura arquitectónica descrita en el *Capítulo 6* no se ve afectado ni alterado, y ni tan sólo es consciente de la integración de las nuevas componentes. En lo sucesivo se utiliza el término "ControladoraObrasAumentada" para hacer referencia a la herramienta resultante de acoplar el *Motor de Plasticidad Implícita* en el sistema de partida ("ControladoraObras").

7.1.2.1. Aspectos de apoyo al trabajo en grupo

Con objeto de proporcionar *consciencia de grupo* entre los distintos trabajadores, el *Motor de Plasticidad Implícita* incorpora componentes específicas para su construcción y mantenimiento, como recopilación de la información asociada a las interacciones del trabajador con sus colegas, así como del estado de su propia actividad. Esta información resulta de utilidad sobretodo de cara a construir el *conocimiento compartido* en el *servidor de plasticidad* (consúltense *Capítulos 2*, 4y 6).

Por lo tanto, esta componente proporciona la capacidad para monitorizar las interacciones entre los miembros del grupo, así como para disparar ciertas acciones automáticas asociadas a cierto tipo de incidencias, con objeto de compartir información relacionada con el estado de las tareas en curso a lo largo de su ejecución. El objetivo es mecanizar la notificación de ciertos eventos a los miembros implicados, los cuales pueden ser de utilidad en el desarrollo de la actividad colaborativa, pudiendo incluso llegar a alterar la actividad prevista para la jornada de algunos de los trabajadores, como consecuencia de ciertos reajustes en sus respectivas agendas. Estos mecanismos, puestos en práctica en segundo plano, constituyen los *mecanismos locales de consciencia de grupo* (véase *Definición 2.2*; *Capítulo 2 - sección 2.1.3.*).

La meta perseguida es la de mejorar la efectividad del grupo de trabajo tratando de fomentar la colaboración.

7.1.2.2. Aspectos de personalización al usuario

El objetivo del *Motor de Plasticidad Implícita* es el de proporcionar adaptaciones dinámicas a ciertas *restricciones de tiempo real* establecidas en la fase de diseño, entre las cuales pueden incluirse ciertos parámetros relativos al patrón de actuación del usuario. En este último caso se trata de la habilidad de inferir las preferencias del usuario a partir de

la observación de la propia ejecución, habilidad propia de los *sistemas adaptativos* (véase *Capítulo 2*; *sección 2.1*).

Los casos de estudio tratados se centran en la inferencia de ciertas preferencias en torno al patrón de actuación del usuario, observado en puntos muy concretos de la ejecución del sistema. Aplicando ciertas heurísticas sobre las pautas observadas se obtienen ciertas deducciones, las cuales son tomadas en consideración a la hora de presentar ciertos aspectos de la IU de forma personalizada. El objetivo perseguido es el de facilitar la realización de acciones interactivas, tales como la selección de uno o varios ítems en una lista determinada, o el cumplimentar un formulario de uso habitual.

7.1.2.3. Aspectos de adaptación al entorno y a las restricciones hardware

Entre las *restricciones de tiempo real* a ser tratadas por el *Motor de Plasticidad Implícita* están las relativas al entorno que rodea la actividad. Esta componente, relacionada con el concepto de *sensibilidad al contexto*, adquiere verdadera relevancia en entornos asociados a cierta movilidad.

Por otro lado, el hecho de recurrir a un dispositivo móvil introduce una serie de limitaciones importantes que pueden llegar a ser críticas en determinadas situaciones para la completitud de las tareas con normalidad. Se trata de aspectos relacionados con las restricciones hardware, como por ejemplo el nivel de batería, el nivel de conectividad en red, el nivel de memoria disponible, etc. La monitorización de recursos hardware de naturaleza dinámica –en particular los denominados *aspectos tecnológicos de naturaleza dinámica* descritos en el *Capítulo 2*; *sección 2.1.3.*- y la consecuente detección y tratamiento de ciertas situaciones críticas o de baja confortabilidad para el usuario constituye también una de las posibles funcionalidades a ser embebidas en el sistema a través del *Motor de Plasticidad Implícita*. No obstante, este aspecto no ha sido considerado en los casos de estudio trabajados.

La reacción del sistema ante una variación en el entorno o en las restricciones hardware puede ser muy diversa dependiendo de la relevancia del cambio o de la dificultad para la completitud de la actividad, dadas las circunstancias. Habitualmente se refleja en la IU, aunque en algunas ocasiones, en general al tratarse de restricciones importantes en los recursos hardware, puede implicar ciertas variaciones en la funcionalidad del sistema subyacente.

7.1.3. Diseño del Motor de Plasticidad Implícita por componentes

Tal y como se plantea en la descripción del caso de estudio, este sistema está planteado como aplicación a ser ejecutada en un teléfono móvil. La plataforma escogida es la modalidad J2ME[1] de Java [Piroumian, 02].

Con objeto de describir las distintas componentes se utiliza de forma genérica el término *entidad objetivo* para hacer referencia a la entidad conceptual manejada por la funcionalidad del sistema base, representada a través de una clase en la *capa lógica*.

7.1.3.1. Componente de personalización al usuario

En esta sección se presentan dos objetivos de personalización distintos, a incidir sobre dos pantallas distintas de la aplicación de partida "ControladoraObras". Se desarrollan dos componentes distintas con objeto de abordar ambos objetivos por separado.

7.1.3.1.1. Objetivo de personalización A: 'valores por defecto en un formulario'

La utilización de parámetros por defecto en el rellenado de un formulario constituye una práctica habitual en el mundo Web, a efectos de agilizar la labor al usuario. Si se piensa en dispositivos compactos, ofrecer esta facilidad resulta muy interesante, a fin de evitar ciertos inconvenientes en la introducción de los datos, así como un consumo de tiempo excesivo. Es en este aspecto que la introducción de una componente de personalización resulta atractiva, a fin de proponer unos valores por defecto con cierta posibilidad de ser los finalmente aceptados por el usuario. Cuanto más 'acertada' es la combinación de valores que se le presentan a priori al usuario, menor número de accesos a los componentes del formulario es necesario realizar, consiguiendo agilizar la introducción de datos.

Para ello se requiere monitorizar la propia acción de cumplimentar el formulario para capturar los valores introducidos, a los que aplicar una cierta heurística o criterio para inferir la combinación con más posibilidades de ser utilizada. Comúnmente, la heurística aplicada de manera generalizada es la de *'elegir la opción más frecuentemente utilizada'*, mostrando en ese caso lo que se denominará a partir de ahora la *combinación de parámetros de moda*.

[1] Modalidad *Micro Edition* destinada a sistemas de prestaciones limitadas.

Peculiaridades del objetivo de personalización A aplicado a este caso de estudio

Con objeto de caracterizar de manera uniforme el modo en que se aplica un determinado objetivo de personalización en los distintos casos de estudio se detallan una serie de peculiaridades susceptibles de variar de un caso a otro, a fin de comprender las necesidades específicas de cada uno. Este ejercicio constituye un paso apropiado de cara a la construcción de un framework genérico. Cuanto mejor caracterizados estén los casos de estudio más fácil será diferenciar las consideraciones comunes de las específicas.

Formulario a personalizar. El informe correspondiente a una tarea/jornada, al que se hace referencia como *FormularioInforme*. Se introducen los detalles relativos al desarrollo de una *tarea* una vez finalizada. Si al acabar la jornada laboral la tarea en la que se está trabajando no ha podido ser completada, se utiliza este mismo informe para detallar la labor realizada a lo largo de la jornada de trabajo.

Heurística empleada. Se ha escogido la de *'elegir la combinación de parámetros más frecuentemente utilizada'*, esto es, la combinación 'de moda'.

Amplitud del muestreo. La amplitud del muestreo que se cree conveniente en este caso para el cálculo de la moda abarca tan sólo una jornada laboral.

Otras particularidades. En la comparación entre informes tan sólo intervienen aquellos parámetros que tienen mayor probabilidad de repetirse. En este caso son los valores para el personal y el tipo de material utilizado.

En definitiva, se propone integrar en el sistema una componente de personalización encargada de capturar las combinaciones de parámetros introducidas por el usuario en el *informe de una tarea/jornada* a lo largo de la jornada de trabajo y registrarlas, a fin de averiguar y mantener en todo momento la *combinación de parámetros de moda* correspondientes a una sesión. Al finalizar la jornada la *combinación de parámetros de moda* se registra de manera persistente, con objeto de que al día siguiente se disponga de una combinación de referencia con la que comenzar.

7.1.3.1.2. Componente para el objetivo de personalización A

A continuación se detallan las unidades *aspecto* y las clases que intervienen en el diseño de esta componente, las cuales se describen por capas. Se identifican también las clases de la capa lógica que intervienen.

Clases de la capa lógica involucradas

La pantalla objeto de personalizacización, que en este caso se trata de un formulario, al que se hace referencia como *FormularioInforme*. La clase *Informe*, como *entidad objetivo* es, por supuesto, otra de las clases afectadas.

Por último, otra clase involucrada es la *clase principal*, puesto que es la pieza de código que inicia y finaliza la ejecución, instantes en los que se procede a cargar en memoria y almacenar de nuevo persistentemente una tabla con las combinaciones de parámetros recogidos. Esta clase se denomina *ControlObrasMIDlet*.

Capa sensible al contexto

La *capa sensible al contexto* agrupa todo lo concerniente a la captura y representación de las *restricciones de tiempo real* a tratar, las cuales se representan a través del denominado *modelo contextual*. Los datos que conforman el *modelo contextual* para el objetivo de personalización objeto de estudio son los relativos a todas las combinaciones de parámetros introducidas por el usuario, a fin de determinar cuál es el '*Informe de moda*'. Para soportar esta información y ofrecer la funcionalidad descrita se incorporan dos clases, las cuales se describen a continuación.

Clase 'UsoInforme'

Objetivo: Estructura de datos para representar la información relativa al número de ocurrencias de cada informe presentado.

Clase 'TablaUsosInformes'

Objetivo: Se trata de una representación en memoria de los objetos de la clase *UsoInforme* –estadísticas de las ocurrencias de cada informe- recogidos a lo largo del periodo de *amplitud de muestreo* considerado. La representación elegida para este fin es la utilidad tabla de *hash*[2], por su facilidad y eficiencia en la realización de las operaciones de búsqueda e inserción.

Atributos: Esta *clase* se compone de la tabla de *hash* propiamente dicha, el *Informe de moda* (la *combinación de parámetros de moda*) y el contador correspondiente

Métodos: Los métodos proporcionados por esta clase son: (a) *cargarTabla* (carga en memoria el historial de los informes, a la vez que establece cuál es el *Informe de moda*); (b) *almacenarTabla* (almacena la tabla de forma persistente) -en el caso particular que nos ocupa tan sólo se registra la *combinación de parámetros de moda* por las razones mencionadas-; y (c) *actualizarOcurrencia* (actualiza la tabla cada vez que el usuario cumplimenta un informe).

[2]Del paquete *java.util*. Estructura de datos por excelencia para mapear claves con valores.

Capa aspectual

En general, las unidades *aspecto* que se requieren incluir como componentes de la *capa aspectual* por lo que respecta a un determinado objetivo de personalización son dos, a los que se les ha denominado *aspecto IncrementadordeOperaciones* y *aspecto CapturadorAdaptador*. A continuación se presentan en detalle.

Aspecto IncrementadordeOperaciones

Como se intuye de su identificador, el *aspecto IncrementadordeOperaciones* introduce nuevos métodos en la estructura de *clases* del código base, permitiendo modificar la estructura estática del programa. Dichos métodos son invocados por el *aspecto CapturadorAdaptador* -código correspondiente a los *consejos*. Estos métodos se incorporan utilizando *definiciones inter-tipo* (véase *Apéndice A*). En concreto, incorpora un total de cuatro métodos:

Métodos a incorporar en la *clase FormularioInforme*:

1. *obtenerInforme*. Se encarga de capturar el informe que acaba de ser introducido por el usuario, justo antes de iniciar el curso normal de su procesamiento.
2. *establecerInformeporDefecto*. Establece como valores por defecto la *combinación de parámetros de moda*.

Métodos a incorporar en la *clase Informe*, con objeto de completarla con las operaciones relacionadas con la gestión de la tabla de *hash* (*clase TablaUsosInformes*) y el cálculo de la combinación de parámetros de moda. Se trata de los dos métodos *equals* y *hashCode*.

Aspecto CapturadorAdaptador

Aspecto que gobierna el tratamiento de la funcionalidad de personalización al usuario.

Las funciones de las que se encarga son:

(a) monitorizar la actuación del usuario y adaptar la IU en base al patrón inferido. Se requieren dos intercepciones en la ejecución con dos propósitos diferenciados: el de captura de datos y el de adaptación. Para ello se vale de los métodos introducidos por el *aspecto IncrementadordeOperaciones*.

(b) materializar y serializar el historial de informes al inicio y en la finalización de la ejecución respectivamente (invocación a los métodos *cargarTabla* y *almacenarTabla* de la clase *TablaUsosInformes*).

Representación del MPI para esta componente

En la figura 7.1 se muestra el diagrama de clases que representa el MPI resultante.

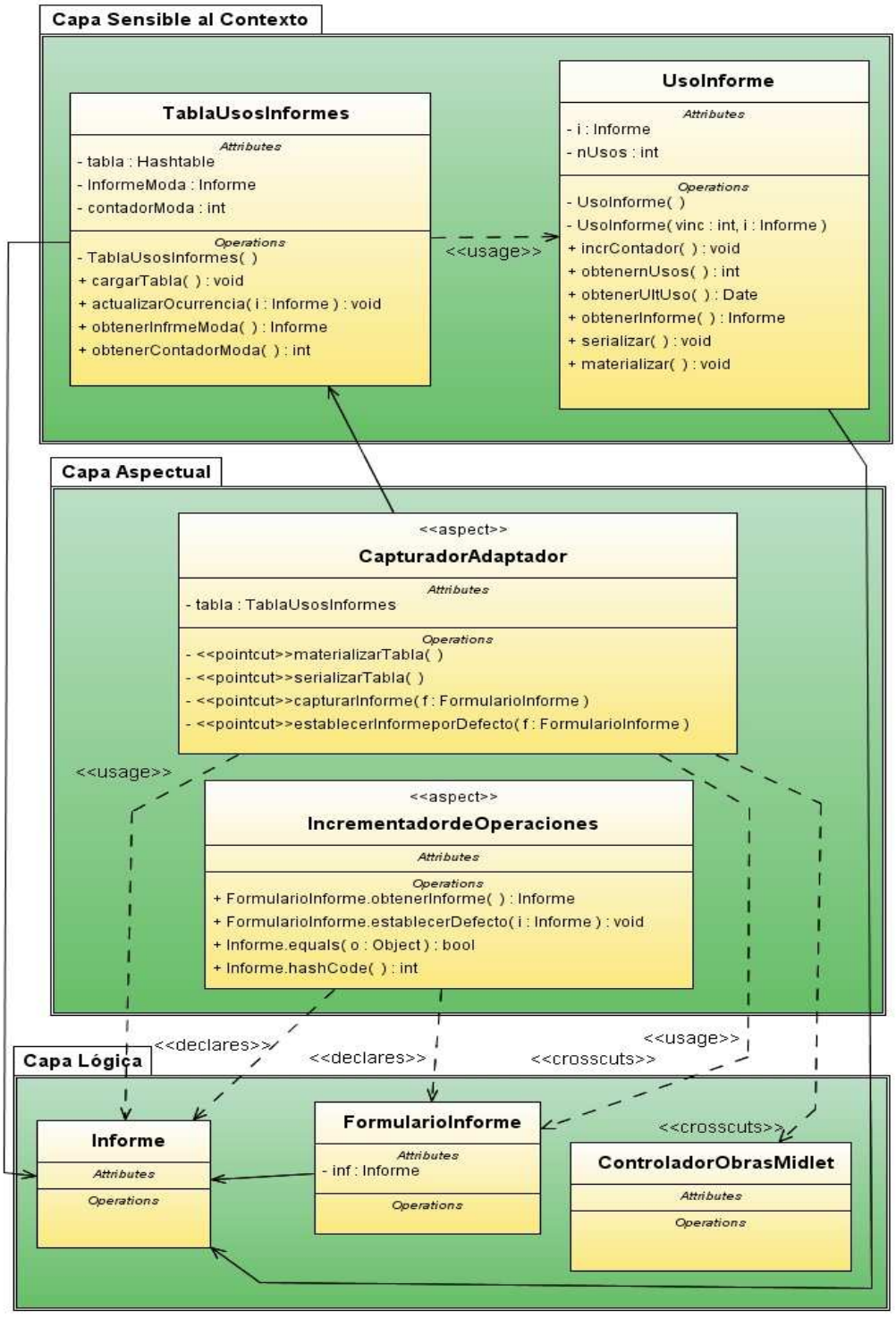


Figura 7.1: Motor de Plasticidad Implícita para objetivo de personalización A (*ControladorObras*).

7.1.3.1.3. Objetivo de personalización B: 'ordenación de una lista'

Es habitual en las aplicaciones cumplimentar el valor de ciertas componentes de un formulario mediante el uso de una lista. Esta acción, que no tiene ningún tipo de dificultad en las plataformas convencionales, puede llegar a ser dificultosa en dispositivos móviles cuando la lista empieza a ser considerable, dadas las limitadas dimensiones de pantalla en estos dispositivos, así como las restricciones en los mecanismos de entrada, de cara a navegar por la lista.

Con objeto de facilitar la selección a los usuarios móviles, una opción es mostrar la lista ordenada de acuerdo a las preferencias o patrón de actuación del usuario, conforme a una cierta heurística. Mostrar la lista ordenada en base al patrón de actuación del usuario supone incrementar las posibilidades de encontrar los valores deseados en las primeras posiciones de la lista.

Peculiaridades del objetivo de personalización B aplicado al caso de estudio

A continuación se detallan las peculiaridades de esta componente.

Lista a personalizar. Uno de los componentes a introducir en el *Informe Tarea/Jornada* tratado en la componente anterior (*FormularioInforme*) es el tipo de material empleado, para el que puede ser necesario destinar varias entradas del formulario. El usuario puede optar por (a) teclear el nombre del material directamente en esta misma pantalla; o (b) seleccionar el material de la lista de material disponible.

Para poder optar por esta segunda opción, el sistema ofrece la posibilidad de acceder a una pantalla tipo lista en la que se muestra la relación de todos los materiales ordenados por nombre. La clase *ListaMaterial* representa esta lista.

De lo que se trata es de personalizar *ListaMaterial* de manera que aparezca ordenada conforme al patrón de actuación del usuario, y de acuerdo a una cierta heurística.

Heurística empleada. La heurística seleccionada es la de *'ordenar decrecientemente según el número de ocurrencias'*.

Amplitud del muestreo. En este caso parece tener sentido recoger el historial de las ocurrencias de material durante un cierto periodo de tiempo (unos días). Por ello es importante en este caso registrar también la fecha de uso.

Otro aspecto a controlar es el de evitar que el registro de usos de material llegue a ser tan grande como el propio archivo de material, dada la frecuencia de esta operación.

En definitiva, se propone integrar en el sistema una componente de personalización encargada de capturar los materiales introducidos a lo largo de los distintos informes cumplimentados por el usuario a través de la pantalla *FormularioInforme*, manteniendo la información acerca del número de ocurrencias de los mismos, a efectos de presentarle al usuario la lista de material ordenada cada vez que accede a la lista. Al finalizar la sesión con el sistema se registra de manera persistente toda la información relativa a los usos de material.

7.1.3.1.4. Componente para el objetivo de personalización B

A continuación se detallan los aspectos más importantes del diseño de esta componente.

Clases de la capa lógica involucradas

El objetivo de personalización planteado en este caso afecta a dos pantallas del sistema base. Una es la pantalla *FormularioInforme*, a la que se accede al cumplimentar un informe al finalizar cada tarea, así como al finalizar la jornada de trabajo. La otra es de tipo *List* denominada *ListaMaterial*.

Se intercepta también la clase principal, denominada *ControladorObrasMidlet*, por los mismos motivos expuestos para la componente anterior. Por último, la *entidad objetivo* que en este caso se trata de la entidad *Material*.

Capa sensible al contexto

Los datos que conforman el *modelo contextual* en este caso son los relativos a los materiales utilizados por el usuario al cumplimentar informes. Esta capa se compone de las dos mismas *clases* utilizadas para el objetivo de personalización anterior, particularizadas a las necesidades y a la *entidad objetivo* aquí manejada.

Clase 'UsoMaterial'

Objetivo: Estructura de datos para representar la información relativa al número de ocurrencias de cada material utilizado.

Una diferencia respecto a la clase *'UsoInforme'* anterior es la inclusión del atributo de la fecha, puesto que interesa almacenar persistentemente no uno, sino varios materiales.

Otra novedad respecto a aquella clase es que *UsoMaterial* implementa la *interfaz Comparable*, con objeto de poder aplicar la ordenación.

Clase 'TablaUsosMateriales'

Objetivo: Se trata de una representación en memoria a través de una tabla de *hash* de los objetos de la clase *UsoMaterial* anterior recogidos a lo largo del periodo de *amplitud de muestreo* considerado.

En este caso se compone de un único atributo: la tabla de *hash*.

A los métodos mencionados anteriormente para la clase *TablaUsosInformes* se incorpora un cuarto método, específico para el objetivo de personalización: *obtenerArrayOrdenado*, a fin de preparar la información de la tabla para su ordenación. La secuencia ordenada es la que se utiliza para construir la lista a mostrar en la pantalla *ListaMaterial*.

Otras clases que intervienen en esta capa, también específicas para el tratamiento de ordenación a aplicar en esta componente son las siguientes:

- *clase 'QuickSortDecreciente'*. Aplica el algoritmo de ordenación *QuickSort*.
- *interfaz 'Comparable'*. Los objetos de la clase *UsoMaterial* deben implementar esta interfaz.

Ha sido necesario implementar estas nuevas piezas de código debido a que la modalidad de Java J2ME, a diferencia de la J2SE, no ofrece algoritmos genéricos de ordenación en su biblioteca estándar.

Capa aspectual

La misión de la *capa aspectual* en este caso de estudio es la de mostrar al usuario la lista de material personalizada, tal y como ha sido especificado, en la pantalla *ListaMaterial*. Ello implica capturar todos los materiales utilizados en los informes cumplimentados, actualizar la tabla y mantener la lista ordenada.

Igual que en el otro objetivo de personalización, se requieren dos unidades *aspecto*.

Aspecto IncrementadordeOperaciones

En este caso no ha sido necesario incorporar nuevas operaciones a la clase *FormularioInforme*, puesto que ya disponía de los métodos que permiten la obtención y la modificación de los valores de material en el informe.

Se incorporan tan sólo los métodos *hashCode* y *equals* en la entidad objetivo *Material*.

Aspecto CapturadorAdaptador

Al igual que para el objetivo de personalización anterior, las funciones de las que se encarga son:

(a) monitorizar la actuación del usuario y adaptar la IU en base al patrón inferido. Se requieren en este caso tres intercepciones en la ejecución: (1) capturar cada uno de los componentes del formulario relativos al material empleado en el formulario *FormularioInforme* -en este caso se requiere capturar tantos ítems por separado como componentes de material se introducen en cada informe, introduciendo por tanto tantos *puntos de corte* como componentes del formulario destinadas a materiales-; (2) adaptar la lista cada vez que se accede a ella, a fin de mostrar los valores ordenados de la tabla de *hash*; y (3) interceptar el método *commandAction*[3] manejado en esta clase, cada vez que se selecciona una opción de la lista (acción también de adaptación).

(b) materializar y serializar la relación de materiales utilizados por el usuario al inicio y en la finalización de la ejecución respectivamente (invocación a los métodos *cargarTabla* y *almacenarTabla* de la clase *TablaUsosMateriales*).

Representación del MPI para esta componente

En la figura 7.2 (pág. siguiente) se muestra el diagrama de clases que representa el MPI resultante.

7.1.3.2. Componente de apoyo al trabajo en grupo

En esta sección se presentan en detalle tanto los objetivos propuestos para la componente de apoyo al trabajo en grupo como la descripción de la propia componente.

7.1.3.2.1. Escenarios planteados

En este caso, a efectos de caracterizar los objetivos propuestos se procede a describir un pequeño conjunto de escenarios en los que se pone de manifiesto que la automatización de ciertas acciones en situaciones clave que implican diferentes trabajadores puede contribuir a una mayor eficiencia en la consecución de la actividad grupal, como resultado de una mayor colaboración entre los trabajadores.

Escenario 1: Agotamiento del material

La componente de apoyo al trabajo en grupo supervisa el nivel de stock del material advirtiendo de manera automática al departamento de administración de la necesidad de reponer material, liberando al usuario de tener en cuenta este tipo de consideraciones.

[3]Es el método mediante el cual el gestor de aplicaciones notifica al manejador de eventos (interfaz *CommandListener*) la ejecución de un comando de pantalla.

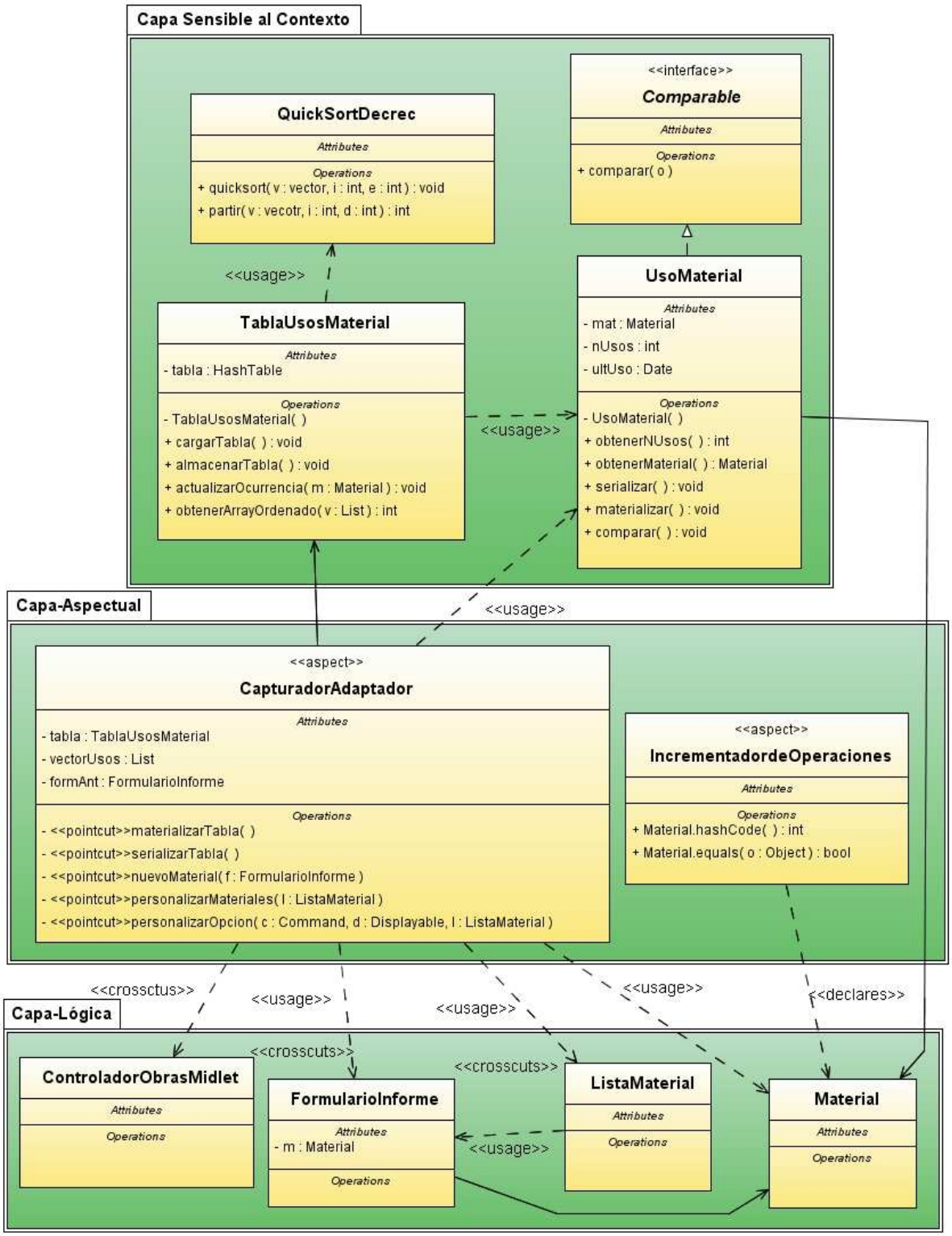


Figura 7.2: *Motor de Plasticidad Implícita* para objetivo de personalización B (*ControladorObras*).

Como cada día, el Sr. González, que es el encargado de su equipo toma cierto material del almacén de la empresa antes de partir hacia las obras. Esta mañana, tras retirar los diez sacos de cemento porlan previstos para su jornada laboral observa que el número de existencias se ve reducido considerablemente, aunque no es consciente de si ha llegado a ponerse por debajo del umbral establecido o no. Por si acaso, anota en su móvil esta incidencia para que en un momento u otro el departamento de administración reciba el aviso oportuno.

Resulta útil controlar este tipo de circunstancias y gestionar el envío de un e-mail de forma inmediata por parte de la componente de apoyo al trabajo en grupo, sin que sea necesaria la intervención explícita del usuario. Se trata de un mensaje predefinido –tan sólo se requiere añadir el material del que se trata- dirigido al departamento de administración, a enviar automáticamente tan pronto como se detecta la situación de escasez de existencias de un cierto material, con el fin de que se tomen las medidas oportunas.

El tratamiento de esta situación en el sistema de partida consiste en anotar esa circunstancia como una incidencia a ser comunicada al final de la jornada, lo que comporta un retraso considerable en su gestión.

Este escenario constituye un ejemplo de *colaboración automática.*

Escenario 2: Cancelación de una tarea.

En este caso se ilustra como la componente de apoyo al trabajo en grupo actúa tomando las acciones oportunas a efectos de facilitar una reorganización inmediata de las agendas de los trabajadores implicados cuando se produce la cancelación de una tarea.

Cada día las agendas de los trabajadores son susceptibles de sufrir cambios debido a ciertas incidencias inesperadas. En este caso, el Sr. González observa que cuando se dispone a restaurar la fachada de la vieja factoría, el andamio no se encuentra montado por completo. En vista de las circunstancias, decide cancelar la tarea que tenía asignada para entonces.

No es suficiente con que el trabajador que ha detectado la incidencia decida pasar a la siguiente tarea planificada en su agenda. El resto de trabajadores de su equipo, implicados en esa misma tarea, no serían advertidos de ese cambio con la debida antelación, de cara a evitar desplazamientos innecesarios. Una notificación de cancelación de tarea a tiempo posibilita una reorganización inmediata de la jornada de trabajo a todos los trabajadores implicados.

Con objeto de proporcionar una coordinación apropiada con el resto de trabajadores es deseable notificar de ese cambio lo antes posible. Una posibilidad es mediante el envío de notificaciones vía SMS, a emitir automáticamente por parte de la componente de apoyo al trabajo en grupo.

En el sistema de partida esta notificación no está contemplada. La responsabilidad de advertir a los demás recae en los propios trabajadores quienes deben contactar explícitamente con sus compañeros de equipo uno por uno. La funcionalidad extra a introducir consiste en monitorizar la cancelación por parte de uno de los miembros del equipo, y una vez detectada la situación, automatizar la emisión de SMSs al resto de los componentes del equipo. Se trata de mensajes de texto predefinidos donde tan sólo se requiere añadir la tarea de la que se trata. Paralelamente, debe ser monitorizado también la recepción de mensajes por parte de cualquier trabajador.

Por otro lado, el trabajador que ha detectado la incidencia tiene una nueva perspectiva de la actividad grupal. En efecto, la cancelación de una tarea constituye un evento suficientemente significativo para el grupo, ya que en algún momento deberá reemprenderse esa tarea. Es importante compartir esa nueva situación no sólo con los colegas directos, sino también con el resto de los miembros del grupo, con el propósito de generar un *entendimiento compartido* de la situación grupal.

En este caso el tratamiento de esta situación se vería complementado haciendo intervenir al *Motor de Plasticidad Explícita*, ubicado en el *servidor de plasticidad*, que siguiendo el modelo de la *visión dicotómica* se dispararía mediante el envío de una petición explícita por parte de la plataforma cliente, tal y como se expone en los capítulos anteriores.

El *Motor de Plasticidad Explícita* se encarga de registrar esta incidencia en un repositorio centralizado, especificado a través del *modelo de grupo* (véase *Capítulo 4*; *sección 4.1.2.1.2.*). Tal y como se expone en el Capítulo 4, la meta es construir una perspectiva lo más completa posible del *conocimiento compartido* [Collazos et al., 02] (véase *Capítulo 2*; *sección 2.1.3*). La emisión de la petición desde la plataforma cliente hacia el *servidor de plasticidad* forma parte también de la funcionalidad incorporada por la componente de apoyo al trabajo en grupo.

Este escenario constituye otro ejemplo de *colaboración automática.*

Escenario 3: Finalización de la agenda para la jornada.

En este caso se ilustra cómo la componente de apoyo al trabajo en grupo actúa tomando las acciones oportunas a efectos de dar a conocer al *servidor de plasticidad* el progreso en la realización de las distintas tareas asignadas para la jornada, así como la indicación de en qué momento se encuentra ocioso. Las deducciones e inferencias llevadas a cabo en el *Motor de Plasticidad Explícita* pueden llevar a determinar la asignación de las tareas pendientes de finalizar a parte de los trabajadores inactivos, a fin de alcanzar la completitud de todas las tareas como meta de grupo.

El Sr. González, al concluir su agenda para la jornada selecciona el comando de finalización de la jornada previsto en la aplicación. Dado que todavía no es la hora de terminar, es

consciente de la posibilidad de tener que reanudar alguna tarea previamente cancelada, por lo que espera a recibir la respuesta proporcionada por el sistema.

En efecto, con el fin de determinar si el trabajador en cuestión puede marcharse a casa, o por el contrario se le asigna una nueva tarea, el sistema (para ser precisos, el *Motor de Plasticidad Implícita*) lanza automáticamente una notificación (emisión de una petición) acerca de la situación al *Motor de Plasticidad Explícita*. Por supuesto, esta circunstancia constituye una pieza de información relevante acerca del estado del grupo de trabajo. La petición contiene una descripción detallada de la situación contextual, la cual es convenientemente serializada en formato XML, también de forma automática.

El *servidor de plasticidad* debe inferir las repercusiones derivadas de la situación actual de grupo. En este caso debe deducir si se le asigna a ese trabajador una de las tareas previamente cancelas, o bien se le da el visto bueno para finalizar la jornada de trabajo. Es evidente que para llevar a cabo estas consideraciones y decisiones es indispensable disponer de un *conocimiento compartido* resultante de la recopilación de todas las *consciencias de grupo particular* y notificaciones individuales llevadas a cabo.

Obviamente, este tipo de decisiones deben ser supervisadas por la administración, esto es, se trata de llevar a cabo un proceso semi-automático, tal y como se propone para el *Motor de Plasticidad Explícita* en el *Capítulo 4*. En algunas ocasiones, la toma de decisiones puede incluso ser delegada a un cierto servicio Web.

Una vez la decisión ha sido tomada, el *servidor de plasticidad* actualiza su *conocimiento compartido* y produce una nueva IU con objeto de dar a conocer al usuario la decisión tomada. Si esta decisión implicara a otros trabajadores –otros trabajadores inactivos a los que asignar también la reanudación de la tarea-, se emitirían las oportunas notificaciones.

Este escenario ilustra un ejemplo acerca de cómo el *servidor de plasticidad* gobierna el seguimiento del trabajo en grupo de manera colaborativa, de acuerdo al *conocimiento compartido* y de manera semi-automática, ilustrando un ejemplo de *consciencia de conocimiento compartido*.

Este tipo de funcionalidad extra convierte el sistema móvil inicial, estrictamente individual, "ControladoraObras" en un sistema móvil distribuido colaborativo, provisto de *mecanismos de awareness*; al que se hace referencia como "ControladoraObrasAumentada".

Esta situación sugiere que cada cambio en el contexto de grupo puede ser automáticamente compartido con el resto de grupo, contribuyendo a incrementar las interacciones entre los miembros del grupo y fomentando una mejor colaboración entre los componentes del equipo de trabajo.

7.1.3.2.2. Descripción de la componente

A continuación se detallan los aspectos más importantes del diseño de esta componente.

Clases de la capa lógica involucradas

La funcionalidad descrita para la componente de colaboración afecta a las clases correspondientes a ciertas *entidades objetivo* del sistema base. En este caso se trata de las clases *Tarea*, interceptada por los *aspectos Coordinación* y *Colaboración* y la clase *Material*, interceptada por el *aspecto Comunicación*.

Además, los tres *aspectos* utilizan la clase *Configuración*, una clase que materializa una serie de parámetros que se encuentran almacenados de manera persistente. Este tipo de clases suelen formar parte de paquetes específicos de utilidades de la aplicación base.

Capa sensible al contexto

El *modelo contextual* en este caso consiste en la información relativa a la *consciencia de grupo particular* de un individuo del grupo de trabajo. Se representa a través de la clase *cGrupoPart*. En general, esta clase se encarga de registrar todas las interacciones del trabajador con sus compañeros, que en este caso particular consiste en las notificaciones de agotamiento de existencias al departamento de administración y las notificaciones de tareas canceladas. Además, mantiene el estado en curso del trabajador (activo, pasivo -en espera del consentimiento del servidor para finalizar la jornada- y finalizado) y también el progreso de su actividad (número de tareas pendientes de realizar en cada momento). La clase *cGrupoPart* tiene un periodo de validez que en el caso de estudio tratado corresponde a una jornada de trabajo, lo que significa que debe ser inicializado en cada sesión. Este parámetro depende de cada sistema colaborativo.

Como ya se ha comentado, la clase *cGrupoPart* es actualizada por los *aspectos*, y al mismo tiempo su estado repercute en el tratamiento a llevar a cabo por los mismos, estableciéndose de ese modo una adecuada realimentación.

Además de la clase *cGrupoPart*, la *capa sensible al contexto* se compone de otras piezas de código, encargadas de materializar en el programa cierta información extra necesaria para llevar a cabo una mejor coordinación y colaboración entre los miembros del grupo. En el caso de estudio que nos ocupa se trata de las clases *informacPropia* y *EquipoTarea*. La primera de ellas contiene información al respecto del usuario. La clase *EquipoTarea*, por su parte, contiene toda la información necesaria para conocer quiénes son los compañeros de equipo en la realización de cada tarea, así como la manera como contactar con ellos.

Capa aspectual

En la *capa aspectual* intervienen los *aspectos* de *Comunicación*, *Colaboración* y *Coordinación*, siguiendo las directrices de Ellis en [EGR91]. Cada uno de ellos está especializado en cada una de las tres metas o áreas clave a considerar en una actividad colaborativa para soportar una correcta interacción de grupo. Se trata de fomentar cualquier tipo de interacción usuario a usuario.

Por lo que respecta al *aspecto* de *Comunicación*, de lo que se encarga es de fomentar la emisión de mensajes entre los integrantes del grupo o parte implicada ante determinadas circunstancias. El *aspecto* de *Coordinación* dispara acciones enfocadas a fomentar una mayor colaboración entre los miembros del grupo, a efectos de lograr una mayor fluidez de información en la realización de tareas conjuntas, advirtiendo lo más pronto posible de cualquier tipo de incidencia, como es el caso aquí ante una situación de cancelación de una tarea.

Finalmente, el *aspecto* de *Colaboración* es el que lidera los contactos con el *servidor de plasticidad* (más concretamente con el *Motor de Plasticidad Explícita*), a efectos de centralizar la información relativa a la actividad colaborativa en su conjunto mediante la construcción del *conocimiento compartido*. Este *aspecto* pone al corriente al servidor de las tareas que van siendo finalizadas, así como del momento en que los trabajadores finalizan su jornada laboral. De ese modo, el servidor, que tiene conocimiento también de las tareas canceladas por parte del *aspecto Coordinación*, dispone de toda la información necesaria para inferir ciertas propiedades globales, o incluso tomar ciertas decisiones destinadas a la consecución de la meta de grupo.

Representación del MPI para esta componente

En la figura 7.3 (pág. siguiente) se muestra el diagrama de clases que representa el MPI resultante.

7.1.3.3. Componente de adaptación al entorno

En esta sección se presentan en detalle tanto los objetivos propuestos para la componente de adaptación al entorno como la descripción de la propia componente, detallando las clases y *aspectos* que intervienen clasificados por capas.

7.1.3.3.1. Objetivo planteado: regulación automática del brillo de pantalla

Las aplicaciones para dispositivos móviles están concebidas para ser ejecutadas mientras el usuario se desplaza. Eso implica la posibilidad de atravesar diversas áreas donde las condiciones ambientales pueden estar sujetas a variación, sobretodo si su uso implica

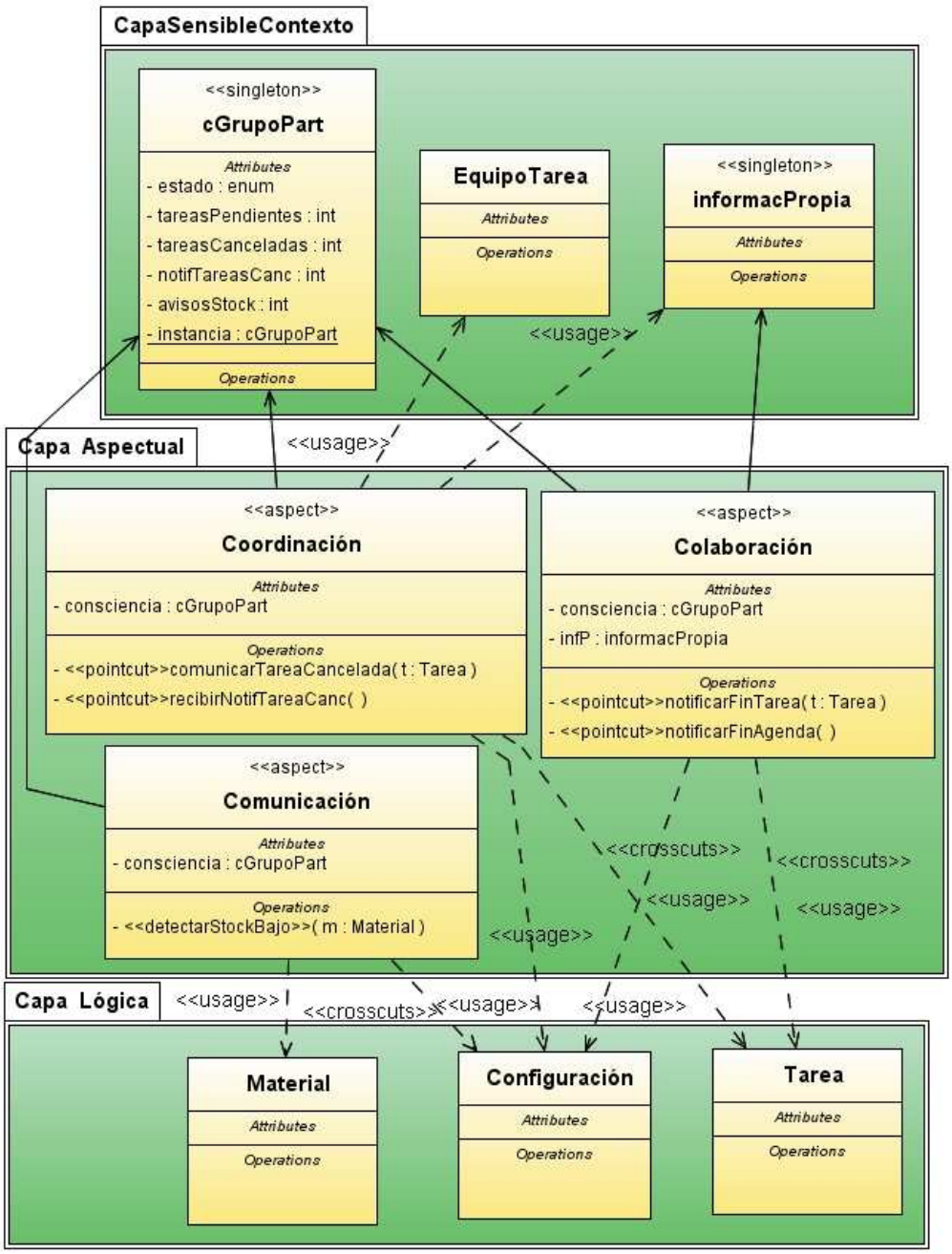


Figura 7.3: *Motor de Plasticidad Implícita* para apoyo al trabajo en grupo (*ControladorObras*).

combinar zonas de exterior con zonas de interior. El sistema propuesto para el caso de estudio está pensado precisamente para ser utilizado en diversas situaciones a lo largo de

la jornada de trabajo, puesto que se trata de una herramienta de asistencia en todas las acciones relacionadas con el desarrollo de las tareas designadas durante la jornada laboral. Podrá ser consultada en diversos lugares (almacén, oficinas, en la furgoneta durante el desplazamiento, en las obras o en el exterior) y bajo diversas condiciones (alta intensidad de luz durante un día soleado en zonas exteriores o baja intensidad, como por ejemplo en las obras sin luz artificial) a lo largo de toda la franja horaria correspondiente a una jornada.

Existen varios factores ambientales que irán variando, pero el que influirá especialmente en la confortabilidad del usuario durante el uso del sistema es el de la luminosidad ambiental. Así, si el sistema es capaz de regular automáticamente el nivel de brillo de la pantalla de acuerdo a las condiciones lumínicas ambientales, el usuario podrá seguir trabajando con cierta confortabilidad sin dejar de estar centrado en lo que está haciendo. En la literatura la sensibilidad a las condiciones lumínicas ha sido aplicada, por ejemplo, en un sistema de seguimiento basado en visión por ordenador [CCB00].

Peculiaridades del tratamiento del la luminosidad aplicado al caso de estudio

A continuación se describe en detalle el propósito y parámetros asociados a esta componente, a fin de identificar posibles variaciones con la componente de adaptación al entorno para el segundo caso de estudio descrito en la sección siguiente.

Factor ambiental a tratar. El nivel de brillo de la pantalla, a regular en función de la intensidad de luz ambiental.

Frecuencia y procedimiento de consulta del sensor. Se aplica este tratamiento (percepción, captura y adaptación) conforme el usuario interactúa con el sistema. Se trata de una aplicación interactiva, que dadas sus características y funcionalidades hace pensar que en los intervalos de tiempo en que el usuario no está interactuando es porque no la está utilizando.

Efectividad de las variaciones. El brillo de pantalla no se regula cada vez que se detecta un cambio, sino tan sólo cuando la variación en la luminosidad externa capturada -o, en su caso, simulada- es apreciable. Se considera que una variación es 'apreciable' cuando la diferencia entre el valor actual captado y el último que provocó un cambio en la pantalla es superior a un cierto umbral.

Otras particularidades. Esta componente se encarga también de registrar los valores que han provocado una alteración del brillo de la pantalla, a efectos de disponer de un histórico que refleje la tendencia experimentada durante la sesión. El histórico es transmitido al *servidor de plasticidad* al finalizar la sesión. De hecho, el *almacenamiento*

del contexto constituye uno de los requisitos de una herramienta de soporte al contexto identificados por Dey [Dey00].

7.1.3.3.2. Componente de regulación automática del brillo de pantalla

A continuación se detallan los aspectos más importantes del diseño de esta componente.

Clases de la capa lógica involucradas

En este caso, tal y como se ha planteado en el apartado 'frecuencia y procedimiento de consulta del sensor', todas las pantallas que integran la aplicación base están involucradas en el tratamiento de la luminosidad exterior, dado que todas implican una interacción con el usuario, ya sea en forma de activación de comandos, la variación de algún ítem o la selección de opciones. Otra clase involucrada es la *clase principal*, como pieza de código encargada de finalizar la ejecución. Precisamente es al finalizar la sesión que se procede a enviar el histórico recogido al *servidor de plasticidad*.

Capa sensible al contexto

La *capa sensible al contexto* en este caso contempla toda la funcionalidad asociada a la consulta de la luminosidad ambiental y consecuente regulación del brillo de la pantalla. Se han desarrollado piezas de código suficientemente cohesivas como para facilitar su reutilización en distintos sistemas.

Se compone de una clase principal, la denominada *GestorLuminosidad*, que desempeña el papel de controladora. Encapsula los objetos *Sensor* y *HistoricoCambios*. Entre otras funcionalidades, la clase *GestorLuminosidad* se encarga de manipular el histórico y de encapsular la comunicación con el sensor.

Las clases auxiliares son: (1) la clase *HistoricoCambios*, que representa la estructura de datos para la confección del histórico de variaciones de la luminosidad ambiental a lo largo de la sesión; (2) la clase *Sensor*, que encapsula todo lo relativo a la comunicación con el sensor o, en su caso, la simulación del mismo; y por último (3) la clase *ReguladorPantalla*, la cual encapsula la adaptación a llevar a cabo ante una variación en el factor ambiental.

Dado que tanto la alteración del brillo de la pantalla como la consulta del valor de un sensor constituyen funcionalidades específicas de APIs propietarias, resulta apropiado encapsular la invocación a las mismas.

Capa aspectual

El hecho de que los *inputs* que conforman el *modelo contextual* provengan en este caso exclusivamente del entorno exterior propicia la independencia entre las capas extremas. Por ello también resulta lógico que no sea necesario introducir nuevos elementos en la jerarquía estática de la aplicación base, con lo que no es necesario contar con el *aspecto IncrementadordeOperaciones*. Es suficiente con una única unidad *aspecto* a la que se le denomina *ControlLuminosidad*.

Aspecto ControlLuminosidad

Aspecto que contiene el código encargado de regular el brillo de la pantalla en este caso de estudio y, en general, de aplicar cualquier tipo de adaptación o ajuste en la pantalla ante un cambio considerado suficientemente significativo en el factor ambiental o en la restricción hardware tratada. Es esta unidad de programa la que dispara las consultas sobre el sensor habilitado, así como la que gobierna la construcción del histórico de muestras obtenidas durante la sesión.

Lleva a cabo, al menos, dos tipos de intercepciones en la ejecución del sistema: (a) al finalizar la sesión de uso del sistema, con objeto de enviar al servidor el histórico de cambios; y (b) a lo largo de la ejecución, a fin de atrapar todos los eventos relacionados con el manejo de la interacción, activando la consulta con el sensor de luz.

Cuando el nivel de interactividad requerida con el sensor es considerable, es adecuado llevar a cabo tanto la detección de los cambios significativos como la ejecución de la correspondiente adaptación a través de una tercera intercepción.

Representación del MPI para esta componente

En la figura 7.4 (pág. siguiente) se muestra el diagrama de clases que representa el MPI resultante.

7.2. El Caso de Estudio *'Lector de Noticias'*

Un diario local ha decidido ampliar los medios utilizados para la difusión de noticias. Con objeto de llegar a un amplio sector de la población se ha optado por implantar un sistema de descarga de noticias a través del uso del teléfono móvil.

La aplicación responsable de gestionar las descargas de noticias ofrecidas por el diario local a un dispositivo móvil de uso personal es la que corresponde a este segundo caso de estudio. Entre otras funciones, ofrece un conjunto de operaciones básicas para la organización y manejo de las noticias descargadas.

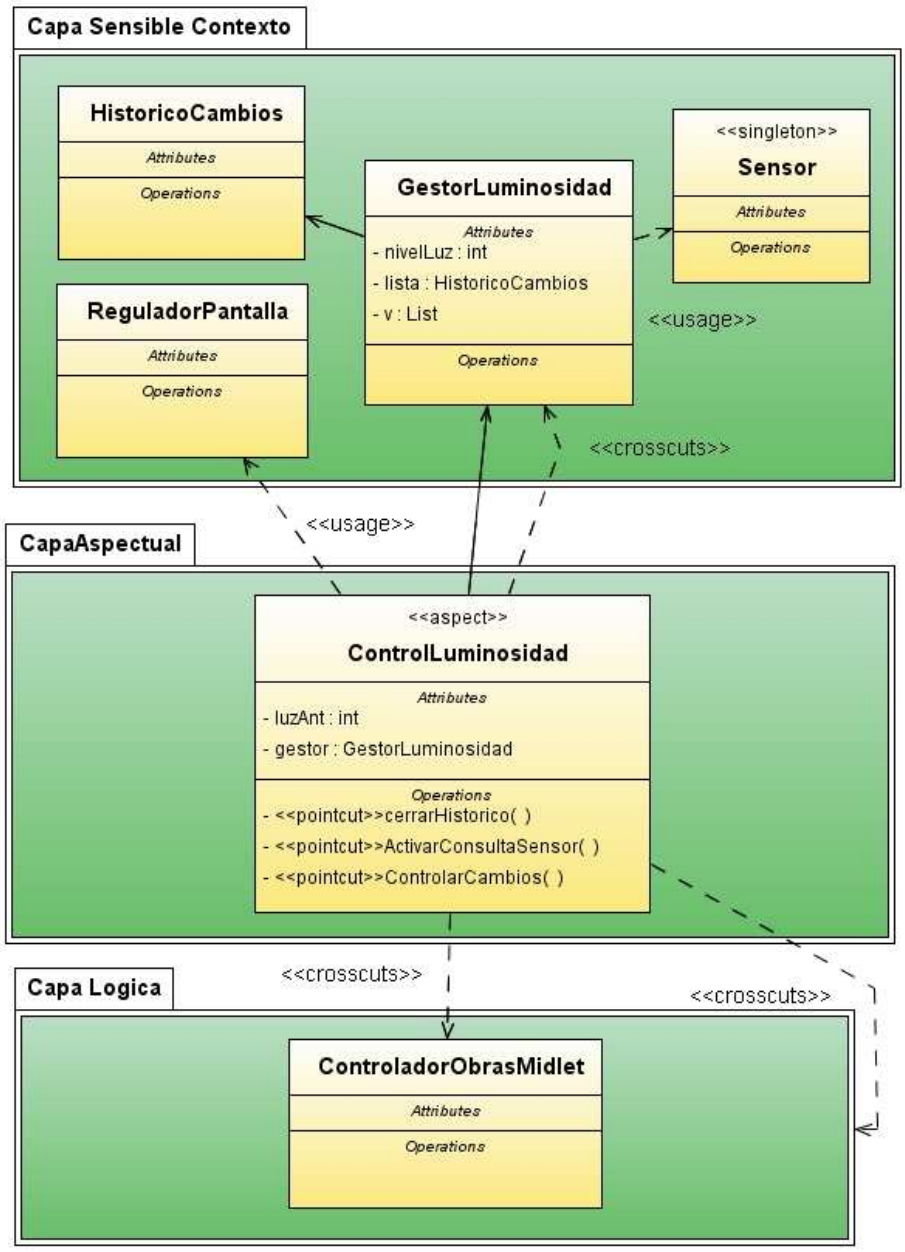


Figura 7.4: *Motor de Plasticidad Implícita* para regular brillo pantalla (*ControladorObras*).

7.2.1. Descripción del sistema de partida

Las noticias ofertadas por el diario local son muy amplias y variadas, pudiendo abarcar temáticas muy diversas. La primera pregunta que cabe realizarse es la de qué noticias desea descargarse cada usuario, puesto que es impensable abarcarlas todas utilizando este

sistema. Para dar respuesta a esta cuestión se introduce el concepto de *perfil de descarga*, a fin de que el usuario establezca sus preferencias. En el perfil de descarga se establecen diversos parámetros como son la temática y el número de días transcurridos desde que se produce la noticia. Además, tiene la posibilidad de seleccionar un conjunto de palabras clave. A partir de entonces la descarga de noticias puede realizarse de manera automática de acuerdo al perfil. En este sentido el sistema se considera un *sistema adaptable* (véase *Capítulo 2*; *sección 2.1.2.*).

No obstante, en ocasiones el usuario puede estar interesado en leer otro tipo de noticias que no corresponden a su perfil preestablecido. El sistema ofrece también la posibilidad de establecer parámetros de descarga específicos para ciertas ocasiones. En lo sucesivo se hace referencia a esta modalidad de descarga como *'descarga manual'*.

Las noticias descargadas, se guardan en el móvil durante un cierto tiempo en una carpeta denominada 'noticias recientes', como almacén para las noticias pendientes de organizar. Los posibles tratamientos sobre las noticias ya descargadas son los siguientes: leer, borrar, enviar vía SMS a un contacto, operaciones de búsqueda y archivar en la carpeta de 'noticias archivadas'. La diferencia entre ambas carpetas –'recientes' y 'archivadas'- es que las 'noticias archivadas' permanecen en el dispositivo hasta que el usuario decide borrarlas. En cambio, las 'noticias recientes' son eliminadas automáticamente una vez transcurrido un cierto tiempo desde su descarga, funcionalidad que se lleva a cabo al finalizar cada sesión. Ese margen de tiempo es establecido por un cierto parámetro.

Esta es la funcionalidad núcleo del sistema de partida, el cual no incorpora capacidades de sensibilidad al contexto, ni tampoco de personalización al usuario. En lo sucesivo se hace referencia al sistema descrito como "LectorNoticias".

7.2.2. Descripción del sistema aumentado

En este caso no es necesario incorporar una componente de apoyo al trabajo en grupo, dado que la herramienta no está concebida para operar en un entorno colaborativo. En efecto, se trata de una aplicación de uso estrictamente personal, por lo que carece de sentido manejar una *consciencia de grupo*.

Se propone aumentar el sistema convirtiéndolo en un *sistema adaptativo*, esto es, capaz de adaptarse a las preferencias y patrones de actuación del usuario en el manejo de ciertas pantallas, facilitando la realización de acciones como la de cumplimentar formularios o seleccionar de una lista, al igual que el caso de estudio previo. Así, por ejemplo, para realizar una *'descarga manual'* mientras el usuario se está desplazando, el esfuerzo que supone tener que cumplimentar todos los parámetros necesarios podría hacer desistir al

usuario. En ese sentido se han desarrollado dos componentes de personalización al usuario: una para adaptar los valores por defecto mostrados en el formulario para la *'descarga manual'*, y la otra para facilitar la búsqueda en la lista de contactos en la tarea de envío de una noticia a un conocido.

Adicionalmente, se destina una tercera componente para personalizar la funcionalidad de borrado de noticias. Así, en función de la frecuencia de uso de la aplicación por parte del usuario y del ritmo de consumo de noticias realizada (descarga y lectura), se trata de ajustar el margen de tiempo que el sistema mantiene las noticias descargadas en la carpeta *'noticias recientes'* al patrón de actuación del usuario.

Como ya se ha mencionado, resulta también especialmente interesante, de cara a proporcionar confortabilidad al usuario en sus desplazamientos, una componente de adaptación al nivel de luminosidad externa, al igual que en el caso de estudio *'Control de Obras'*. No obstante, tal y como se detalla más adelante, esta componente presenta ciertas variaciones, dado que las necesidades no se corresponden completamente a las planteadas para esa otra aplicación.

7.2.3. Diseño del Motor de Plasticidad Implícita por componentes

A continuación se describe una a una cada uno de los componentes a integrar en el sistema "LectorNoticias", a fin de obtener las distintas funcionalidades *sensibles al contexto* y *adaptativas* descritas. En lo sucesivo se utiliza el término "LectorNoticiasAumentado" para hacer referencia a la herramienta resultante de acoplar el *Motor de Plasticidad Implícita* en el sistema de partida ("LectorNoticias").

7.2.3.1. Componente de personalización al usuario

En esta sección se presentan dos objetivos de personalización distintos, a incidir sobre dos pantallas distintas de la aplicación de partida "LectorNoticias".

7.2.3.1.1. Objetivo de personalización A: 'valores por defecto en un formulario'

Igual que en este mismo objetivo de personalización para el caso de *estudio 'Control de Obras'* (véase *sección 7.1.3.1.1.*), la meta a abordar es la de personalizar los valores por defecto de un determinado formulario de la aplicación base.

Peculiaridades del objetivo de personalización A aplicado al caso de estudio

El objetivo de personalización A, o de *'valores por defecto en un formulario'*, hace referencia a la utilidad de facilitar al usuario el rellenado de ciertos formularios proporcionando ciertos valores por defecto, los cuales son inferidos del patrón de actuación del usuario observado durante la ejecución del sistema.

Formulario a personalizar. El formulario utilizado para establecer el perfil de descarga a utilizar en el momento de solicitar una *'descarga manual'*. En lo sucesivo se hace referencia a dicho formulario como *FormularioPerfilManual*.

Heurística empleada. La heurística escogida es la de *'elegir la opción más frecuentemente utilizada'*.

Amplitud del muestreo. Se cree conveniente recoger el historial de perfiles de descarga utilizados durante un cierto periodo de tiempo, a fin de inferir el perfil 'favorito', periodo que se ha estimado de treinta días (ciclos de información de un mes).

Adicionalmente, debe evitarse que el registro de perfiles de descarga llegue a ser demasiado grande, estableciéndose un tamaño máximo para el mismo.

En definitiva, se propone integrar en el sistema una componente de personalización encargada de capturar las combinaciones de parámetros introducidas por el usuario en el perfil para la *'descarga manual'* y registrarlas, a fin de averiguar y mantener en todo momento la *combinación de parámetros de moda* correspondientes a un determinado periodo de tiempo.

7.2.3.1.2. Componente para el objetivo de personalización A

A continuación se detallan los aspectos más importantes del diseño de esta componente.

Clases de la capa lógica involucradas

Las clases afectadas son: (a) el formulario de introducción de los parámetros del perfil de *'descarga manual'*. Se trata de una subclase de *Form* a la que se le hace referencia aquí como *FormularioPerfilManual*; (b) la *entidad objetivo*, que en este caso se representa a través de la clase *Perfil*; y (c) la clase principal, al ser la pieza de código que inicia y finaliza la ejecución, instantes en los que se procede a cargar en memoria y almacenar de nuevo persistentemente la tabla de los usos. Se trata de la *clase LectorNoticiasMIDlet*.

Capa sensible al contexto

Los datos que conforman el *modelo contextual* para el objetivo de personalización propuesto aquí son las combinaciones de parámetros utilizados para establecer el perfil en

una *'descarga manual'*. Para soportar esta información y ofrecer la funcionalidad necesaria para mantener las ocurrencias de cada perfil de descarga utilizado y, en definitiva, poder obtener la *combinación de parámetros de moda*, se incorporan las dos mismas clases utilizadas para este mismo objetivo de personalización en el caso de estudio *'Control de Obras'*, por supuesto adaptadas a la *entidad objetivo* manejada aquí.

Clase 'UsoPerfilDescarga'

Objetivo: representar la información relativa al número de ocurrencias de cada perfil de descarga utilizado.

Clase 'TablaUsosPerfiles'

Objetivo: representar en memoria a través de una tabla de *hash* los objetos de la *clase UsoPerfilDescarga* recogidos a lo largo del periodo de *amplitud de muestreo* considerada.

Los atributos son los que aparecen en la clase *'TablaUsosInformes'* utilizado en este mismo objetivo de personalización para el caso de estudio *'Controlador de Obras'*, además del atributo lógico *cambios*, utilizado para conocer si el usuario ha realizado alguna *'descarga manual'* durante la sesión.

Los métodos son también los mismos: *cargarTabla*, *almacenarTabla* y *actualizarOcurrencia*.

Capa aspectual

Se requieren los dos mismos *aspectos* utilizados en el sistema "ControladoraObrasAumentada", denominados: *aspecto IncrementadordeOperaciones* y *aspecto CapturadorAdaptador*, cuyas responsabilidades son también las mismas.

Aspecto IncrementadordeOperaciones

Métodos a incorporar en la clase *FormularioPerfilManual*:

1. *obtenerPerfilDescarga*. Captura el perfil de descarga que acaba de ser introducido por el usuario, justo antes de iniciar su procesamiento.
2. *establecerPerfilDescargaporDefecto*. Establece los valores del perfil de descarga por defecto a partir de la *combinación de parámetros de moda*.

Métodos a incorporar en la *clase PerfilDescarga*: *equals* y *hashCode*.

Aspecto CapturadorAdaptador

Aspecto que incorpora y gobierna el tratamiento de la nueva funcionalidad de personalización al usuario.

Al igual que para esta misma componente de personalización para el caso de estudio previo, las dos funciones de las que se encarga son:

(a) monitorizar la actuación del usuario y adaptar la IU en base al patrón inferido. Igualmente, se requieren dos intercepciones en la ejecución: para capturar los datos y para adaptar el formulario. Se vale de los métodos introducidos por el *aspecto IncrementadordeOperaciones*.

(b) materializar y serializar el historial de informes al inicio y en la finalización de la ejecución (invocación a los métodos *cargarTabla* y *almacenarTabla* de la clase *TablaUsosPerfiles*).

Representación del MPI para esta componente

En la figura 7.5 (pág. siguiente) se muestra el diagrama de clases que representa el MPI resultante.

7.2.3.1.3. Objetivo de personalización B: 'ordenación de una lista'

En este caso de estudio existe también la necesidad de seleccionar de una lista. Se trata de la lista de *Contactos* proporcionada por el propio dispositivo. Esta componente se encarga de ordenar la lista en base al patrón de actuación del usuario, y de acuerdo a un cierto criterio.

Para poder acceder a la agenda de *Contactos* del móvil desde una aplicación J2ME se utiliza el paquete opcional PIM[4] (*Personal Information Management*).

Peculiaridades del objetivo de personalización B aplicado al caso de estudio

Con el fin de contrastar la solución presentada aquí con la aportada para este mismo objetivo de personalización en el caso de estudio *'Controlador de Obras'*, a continuación se presentan sus peculiaridades.

Lista a personalizar. Una de las opciones que ofrece el sistema "LectorNoticias" es la de enviar una noticia por SMS a un determinado contacto. En el momento de seleccionar la opción *'enviar noticia'*, se le muestra al usuario una nueva pantalla tipo formulario, a la que se le denominará a partir de ahora *PantallaEnviarNoticia*, donde se puede optar por (a) seleccionar el teléfono de la lista de *Contactos* del móvil, evitando tener que teclear los nueve dígitos; o (b) teclear el número de teléfono directamente en esta misma pantalla. Ésta es la opción a la que se recurre, por ejemplo, cuando el teléfono no se encuentra en la lista de *Contactos*.

[4]Paquete opcional que ofrece soporte para acceder y modificar las bases de datos PIM (principalmente la lista de contactos) que alberga el dispositivo MIDP.

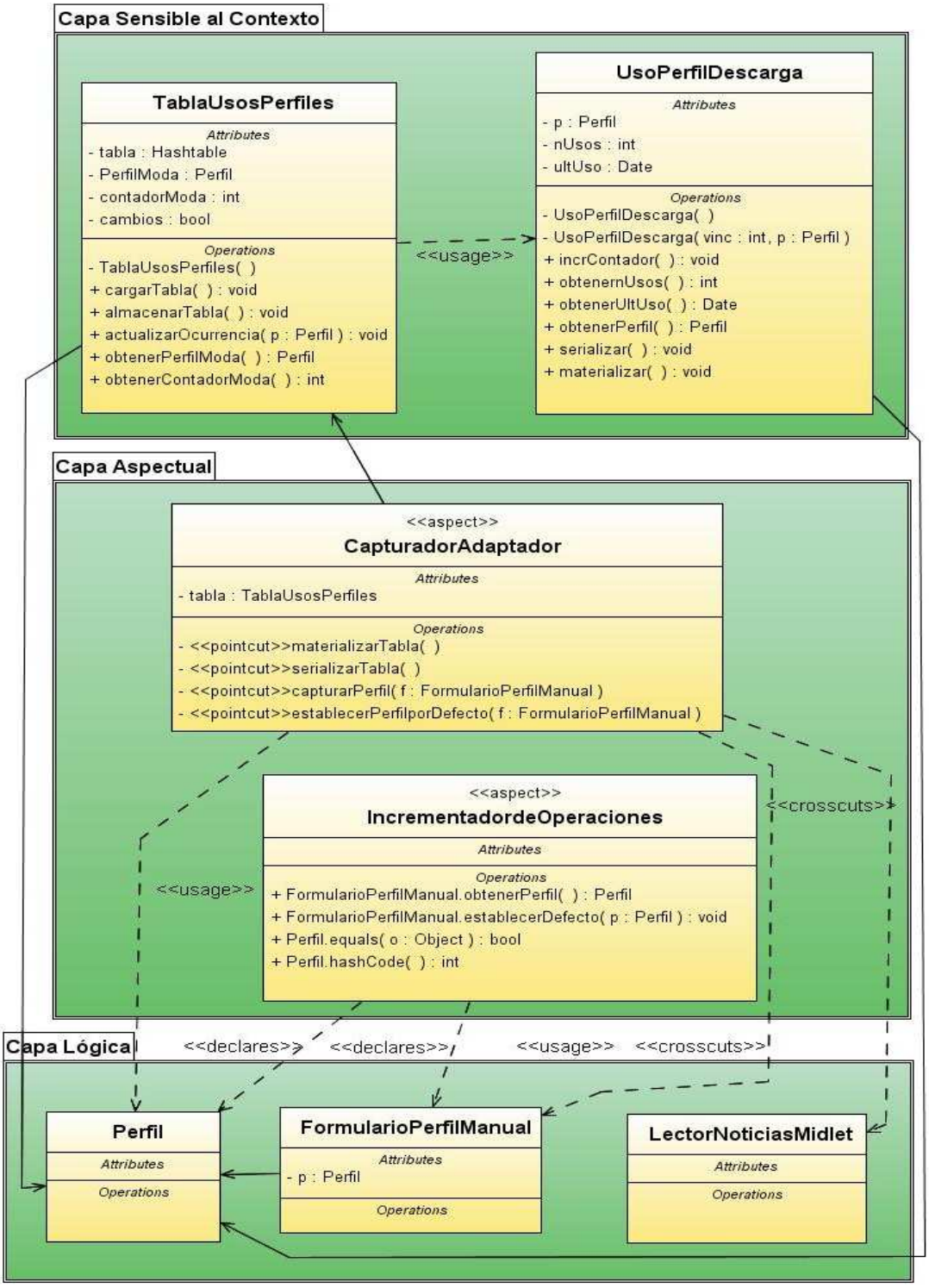


Figura 7.5: *Motor de Plasticidad Implícita* para objetivo de personalización A (*LectorNoticias*).

La primera opción presenta una pantalla tipo lista (*subclase* de *List* en el perfil MIDP) en la que se muestran las entradas a la agenda de *Contactos* del móvil, tal y como si se accediera desde fuera de la aplicación. Se hace referencia a esta pantalla como *'ListaContactos'*.

De lo que se trata es de personalizar esta lista de *Contactos* teniendo en cuenta estas dos consideraciones: (1) la lista debe mostrar no sólo los contactos de la agenda del móvil, sino también cualquier otro número de teléfono previamente utilizado en la realización de esta tarea; y (2) la lista aparecerá ordenada en base a una determinada heurística y conforme a los datos registrados relativos al patrón de actuación del usuario.

Se trata de una personalización de la lista de *Contactos* en contenido y en orden.

Heurística empleada. La heurística seleccionada es la de *'ordenar decrecientemente según el número de ocurrencias'*.

Amplitud del muestreo. Se establece un tamaño máximo para el *RecordStore*. Una vez alcanzado ese umbral, el almacenamiento persistente acumulado se desecha, lo que significa volver al punto de partida, disponiendo tan sólo de los contactos de la agenda. Este es, en este caso, el único criterio para establecer la *amplitud del muestreo*.

En definitiva, se propone integrar en el sistema una componente de personalización encargada de capturar los contactos utilizados por el usuario a lo largo de las distintas sesiones de uso del sistema, y de mantener el conocimiento del número de ocurrencias de los mismos, a efectos de ordenar dinámicamente la lista de *Contactos* de la manera considerada más satisfactoria para el usuario, en la acción de enviar una noticia.

Al finalizar la sesión de uso del sistema se registra de manera persistente toda esa información, con el fin de trabajar con información acumulada de varias sesiones.

7.2.3.1.4. Componente para el objetivo de personalización B

A continuación se detallan los aspectos más importantes del diseño de esta componente.

Clases de la capa lógica involucradas

El objetivo de personalización planteado afecta a dos pantallas del sistema base. Una es la *PantallaEnviarNoticia*, a la que se accede al seleccionar la opción de enviar una noticia. Desde este formulario se accede a la lista de *Contactos* del móvil a través de la pantalla de tipo *List 'ListaContactos'*.

Al igual que en los objetivos de personalización para el sistema "ControladoraObrasAumentada", también está involucrada la clase principal. Se trata de la clase *LectorNoticiasMIDlet*.

La clase que representa la *entidad objetivo*, la entidad *Contacto*.

Capa sensible al contexto

Los datos que conforman el *modelo contextual* son los relativos a las entidades objetivo *Contacto* utilizados por el usuario a lo largo de las sesiones de uso del sistema.

Se incorporan de nuevo las dos mismas clases para representar los usos y la tabla de usos, por supuesto particularizadas a las necesidades aquí planteadas.

Clase 'UsoContacto'

Objetivo: representar la información relativa al número de ocurrencias de cada contacto utilizado.

Esta clase implementa la *interfaz Comparable*, a fin de proceder a la ordenación.

Clase 'TablaUsosContactos'

Objetivo: soportar en memoria los objetos de la clase *UsoContacto* correspondientes a los contactos utilizados. Se utiliza una tabla de *hash* para su representación.

Los métodos proporcionados por esta clase son los mismos que aparecen para la clase *TablaUsosMateriales* en el objetivo de personalización *'ordenación de una lista'* para el sistema "ControladoraObrasAumentada": *cargarTabla*, *almacenarTabla* y *actualizarOcurrencia* y *obtenerArrayOrdenado*.

Además, al igual que en este mismo objetivo de personalización para el caso de estudio *'Controlador de Obras'* se incorporan la clase *QuickSortDecrec* y la *interfaz Comparable*.

Capa aspectual

La misión de la *capa aspectual* en este caso de estudio es la de mostrar al usuario la lista de contactos personalizada tal y como ha sido especificado, en la pantalla *ListaContactos*. Ello implica capturar todos los números utilizados durante la sesión, actualizar la tabla y mantener la lista ordenada. Intervienen también dos unidades *aspecto*: el *aspecto IncrementadordeOperaciones* y el *aspecto CapturadorAdaptador*.

Aspecto IncrementadordeOperaciones

En este caso no ha sido necesario incorporar nuevas operaciones a la clase *PantallaEnviarNoticia*, puesto que ya disponía de los métodos para realizar las operaciones de obtener el número de teléfono introducido, como parte de la información representada, así como la de asignar esa información, como atributo de la *clase PantallaEnviarNoticia*.

No obstante, sí es necesario completar la *entidad objetivo* con los métodos *hashCode* y *equals*.

Aspecto CapturadorAdaptador

Las dos funciones de las que se encarga este *aspecto* son:

(a) monitorizar la actuación del usuario y adaptar la IU en base al patrón inferido. Se requieren en este caso tres intercepciones en la ejecución: (1) capturar el número de teléfono introducido en el formulario *PantallaEnviarNoticia*; (2) adaptar la lista cada vez que se accede a ella, a fin de mostrar los valores ordenados de la tabla de *hash*; y (3) interceptar el método *commandAction*[5] manejado en esta clase, cada vez que se selecciona una opción de la lista (acción de adaptación).

(b) materializar y serializar la relación de contactos utilizados por el usuario al inicio y en la finalización de la ejecución respectivamente (invocación a los métodos *cargarTabla* y *almacenarTabla* de la clase *TablaUsosMateriales*).

Representación del MPI para esta componente

En la figura 7.6 (pág. siguiente) se muestra el diagrama de clases que representa el MPI resultante.

7.2.3.1.5. Objetivo de personalización C: 'ajuste de un parámetro'

En este caso de estudio se incorpora una tercera componente de personalización al usuario que se encarga de adaptar un cierto parámetro asociado a una funcionalidad concreta del sistema.

Descripción

Se trata de la funcionalidad de vaciar la carpeta *'noticias recientes'* de todas aquellas noticias que, una vez transcurrido el periodo de tiempo establecido desde la fecha de su descarga, permanecen sin leer por parte del usuario. En lo sucesivo se hace referencia a este parámetro de tiempo como *'periodo de margen de lectura'*. Así, esta utilidad elimina las 'noticias obsoletas'. Se activa al finalizar cada sesión de uso del sistema comprobando noticia por noticia si son o no obsoletas, y procediendo en consecuencia.

Se trata de una funcionalidad que resulta de gran ayuda al desechar automáticamente las noticias que con cierta certeza han dejado de tener interés para el usuario, sin que éste deba preocuparse de su eliminación expresa, evitando un consumo de tiempo considerable para llevar a cabo el correspondiente vaciado.

[5]Es el método mediante el cual el gestor de aplicaciones notifica al manejador de eventos (interfaz *CommandListener*) la ejecución de un comando de pantalla.

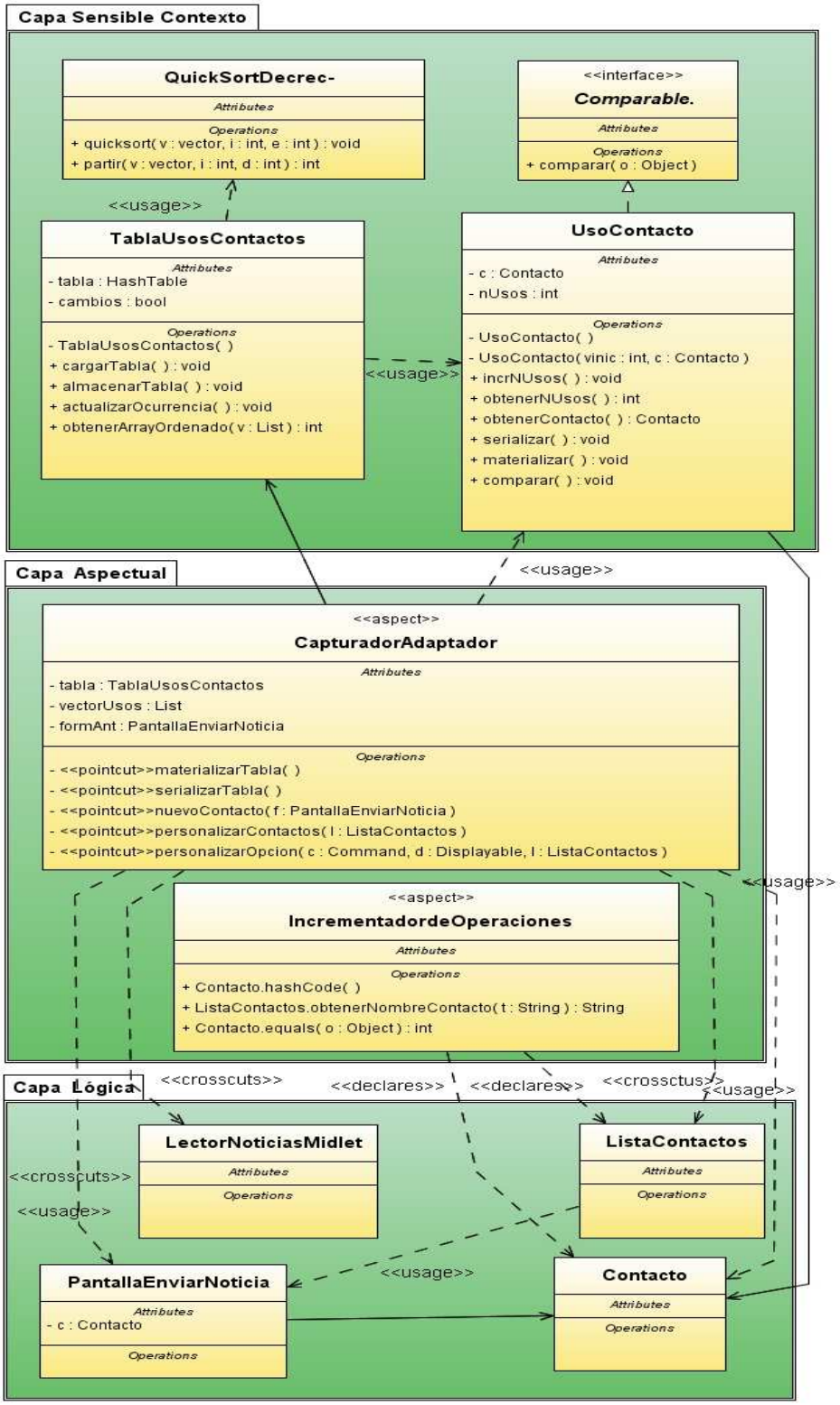


Figura 7.6: *Motor de Plasticidad Implícita* para objetivo de personalización B (*LectorNoticias*).

No obstante, si el parámetro no se ajusta a las necesidades del usuario, esta utilidad puede generar más inconvenientes que beneficios, llegando incluso a producir el efecto contrario al deseado, siendo el usuario quien tiene que adaptarse al funcionamiento del sistema. Si el nivel de satisfacción obtenido por el usuario se aleja del esperado, éste puede acabar abandonando el uso del sistema.

En efecto, si se consideran dos perfiles de actuación contrapuestos: (a) el usuario que utiliza el sistema con mucha frecuencia y consume gran cantidad de noticias diariamente, de manera que las noticias que deja sin leer, si no las ha leído a los pocos días de su descarga es porque ya no las toma en consideración; y (b) aquel que utiliza el sistema con menos frecuencia y, además, pospone la lectura de parte de las noticias que se descarga durante varios días.

Al usuario que responde al perfil (a) esta utilidad le resulta de gran utilidad, y le interesa que el valor del parámetro *'periodo de margen de lectura'* sea relativamente pequeño, pues su carpeta de 'noticias recientes' se renueva continuamente. Si el sistema no lleva a cabo el vaciado con la suficiente brevedad deberá ser el propio usuario quien lo haga, lo cual puede suponer una pérdida de interés importante hacia la aplicación. En cambio, al usuario que responde al perfil (b) no le conviene un valor para el parámetro de tiempo demasiado pequeño, pues con una alta probabilidad perdería parte de las noticias en las que sigue interesado sin aún haberlas leído, por lo que el sistema podría dejar también de serle útil.

Objetivo

En la aplicación base el parámetro *'periodo de margen de lectura'* permanece invariable a lo largo de todas las sesiones y, además, es común para todos los usuarios. La componente *adaptativa* que se ha implementado se encarga de estudiar el patrón de actuación del usuario con objeto de personalizar este parámetro, el cual puede estar sujeto a variaciones a lo largo del tiempo, en base a un determinado periodo de muestreo. Se trata de un ejemplo de *adaptatividad* (véase *Capítulo 2*; *sección 2.1.2*).

En este caso, en lugar de ser la IU la que sufre una cierta modificación o ajuste, la adaptación se aplica sobre una cierta funcionalidad o utilidad.

Peculiaridades del objetivo de personalización C aplicado al caso de estudio

A continuación se caracterizan y especifican ciertas peculiaridades.

Parámetro a personalizar. El parámetro que se desea ajustar de manera dinámica es el denominado *'periodo de margen de lectura'*, de manera que una vez transcurrido

este periodo desde la fecha de descarga de una noticia, ésta es eliminada de la carpeta de 'noticias recientes' si aún permanece sin leer.

Heurística empleada. La heurística más apropiada es la de *'obtener el valor máximo'*, esto es, el tiempo máximo que tarda el usuario en leer una noticia de la carpeta de 'noticias recientes' una vez descargada, de entre todos los valores de tiempo recogidos durante un determinado periodo de muestreo.

Amplitud de muestreo. El periodo de muestreo durante el cual se recogen los tiempos de demora en la lectura de noticias está fijado en quince días. Este es el periodo de observación para conocer el perfil del usuario y establecer el *'periodo de margen de lectura'*.

Otras particularidades. Debe iniciarse el periodo de observación con un valor para el parámetro objeto de estudio por defecto. Éste está establecido en una semana (como promedio del tiempo que se ajustaría a los dos perfiles anteriormente comentados). Transcurridos los quince días de periodo de muestreo ese valor es sustituido por el valor personalizado a cada usuario, conforme a la observación realizada, empezando ya a actuar el nuevo valor.

También se establece un tiempo de demora mínimo, de manera que por debajo de su valor no se considera adecuado establecer el parámetro *'periodo de margen de lectura'*.

Premisas iniciales. Según todo lo comentado, el sistema de partida ya cuenta con la funcionalidad de borrado automático de noticias. Este parámetro se registra de manera persistente en el dispositivo y es materializado al iniciar el programa en una clase que suele formar parte de un paquete de utilidades, a la que se hace referencia aquí como la clase *Configuración*. Este tiempo es fijo para todos los usuarios, y además permanece invariable.

De nuevo, al igual que en los otros objetivos de personalización presentados, es necesario poner en marcha un mecanismo de monitorización en segundo plano, a efectos de estudiar el patrón de actuación del usuario en relación al tiempo que se demora en leer las noticias descargadas, con objeto de inferir un valor para el parámetro *'periodo de margen de lectura'* lo más satisfactorio posible a lo largo del tiempo.

Con objeto de que esta monitorización se pueda efectuar a lo largo de varias sesiones de uso se almacenan estos datos en un archivo persistente al finalizar cada sesión.

7.2.3.1.6. Componente para el objetivo de personalización C

A continuación se detallan los aspectos más importantes del diseño de esta componente.

Clases de la capa lógica involucradas

El tratamiento planteado aquí afecta a la clase encargada de llevar a cabo la funcionalidad objetivo, la cual ya forma parte del sistema base y a la que se hace referencia como '*BorradoNoticias'*. Esta clase utiliza un valor prefijado de antemano como valor para el parámetro *'periodo de margen de lectura'*. Si el sistema viene provisto de una clase que encapsula los parámetros propios de configuración, ésta también es interceptada, a efectos de sustituir el valor preestablecido para el parámetro objeto de personalización por el valor inferido del proceso de monitorización llevado a cabo. Por último, la clase que representa la *entidad objetivo*, que en este caso se trata de la clase *Noticia*.

Capa sensible al contexto

En este caso, el *modelo contextual* se construye de acuerdo al parámetro objeto de personalización. En este caso es representado a través de una única clase denominada '*GestiónParmBorradoNoticias'*. Su responsabilidad es la relacionada con la gestión de los parámetros relacionados con la funcionalidad objeto de estudio, siguiendo las particularidades mencionadas.

Capa aspectual

La misión de la *capa aspectual* en este caso de estudio es muy concisa. Al inicio de la ejecución se materializa el *modelo contextual* registrado de manera persistente y se establece el valor del parámetro *'periodo de margen de lectura'* accediendo a la clase *Configuración*. Interceptando esta clase se sustituye el valor inicial para el parámetro por el valor personalizado.

Por otro lado, está la parte encargada de capturar el tiempo de demora en la lectura de todas las noticias, a fin de obtener el valor máximo, constituyendo la parte de monitorización de esta componente.

La acción correspondiente a la adaptación consiste en interceptar la propia acción de borrado de noticias, ya soportada por una clase del sistema base, acción que se realiza al finalizar la ejecución. Una vez ejecutada esta acción se procede a la actualización efectiva de los parámetros que están en juego (el tiempo de demora a establecer y el inicio de un nuevo periodo de muestreo, si es el caso). A continuación se procede al almacenamiento persistente de dichos parámetros, a punto para el inicio de la siguiente sesión.

La funcionalidad extra es incorporada en la ejecución de la aplicación por parte del *aspecto CapturadorAdaptador*. Dependiendo de los casos, puede ser necesario recurrir al *aspecto IncrementadordeOperaciones* a fin de incorporar los métodos a interceptar del sistema base, siempre y cuando éstos no formen ya parte integrante del mismo. Para el caso que nos ocupa debería garantizarse la presencia del método encargado de marcar la noticia como leída, y la del método para obtener la fecha de descarga de una noticia.

Aspecto CapturadorAdaptador

Para personalizar la funcionalidad objeto de estudio, es necesario tanto capturar ciertos ítems de información en el momento oportuno, como llevar a cabo ciertas acciones de adaptación. Así, en el momento de leer una noticia, se obtiene el tiempo de demora en la lectura de la misma, a fin de determinar el valor máximo para el *'periodo de margen de lectura'* a lo largo del periodo de muestreo establecido. Se destina para ello un *punto de corte* (*capturarComportamiento*). Las acciones de adaptación requeridas son dos: (1) suplantar el valor prefijado por la aplicación base para el parámetro *'periodo de margen de lectura'* por el valor establecido en el *modelo contextual* -una posibilidad sería interceptar el método constructor de la clase base *Configuración*, a través de un *punto de corte*-; y (2) interceptar la propia acción de borrado de noticias de la aplicación base.

Las funciones propias de materialización y serialización de los valores asociados a esta gestión quedan integradas en la funcionalidad descrita.

Representación del MPI para esta componente

En la figura 7.7 (pág. siguiente) se muestra el diagrama de clases que representa el MPI resultante.

7.2.3.2. Componente de adaptación al entorno

En esta sección se presentan en detalle tanto los objetivos propuestos para la componente de adaptación al entorno como la descripción de la propia componente, detallando las clases y *aspectos* que intervienen clasificados por capas.

7.2.3.2.1. Objetivo planteado: regulación automática del brillo de pantalla

La funcionalidad ofrecida por el sistema "LectorNoticias", a ejecutar por un dispositivo móvil, está concebida para su utilización en cualquier lugar y en cualquier momento, incluso cuando el usuario se está desplazando. Como la lectura de las noticias descargadas requiere mantener la vista fijada en la pantalla durante un cierto tiempo, con la posibilidad de atravesar distintas condiciones de luminosidad ambiental, resulta interesante ofrecer la capacidad de autorregular automáticamente el brillo de la pantalla de acuerdo a las condiciones de luz con las que se vaya encontrando el usuario.

Peculiaridades del objetivo de adaptación aplicado al caso de estudio

A continuación se presentan las distintas peculiaridades para el caso de estudio *'Lector de Noticias'*.

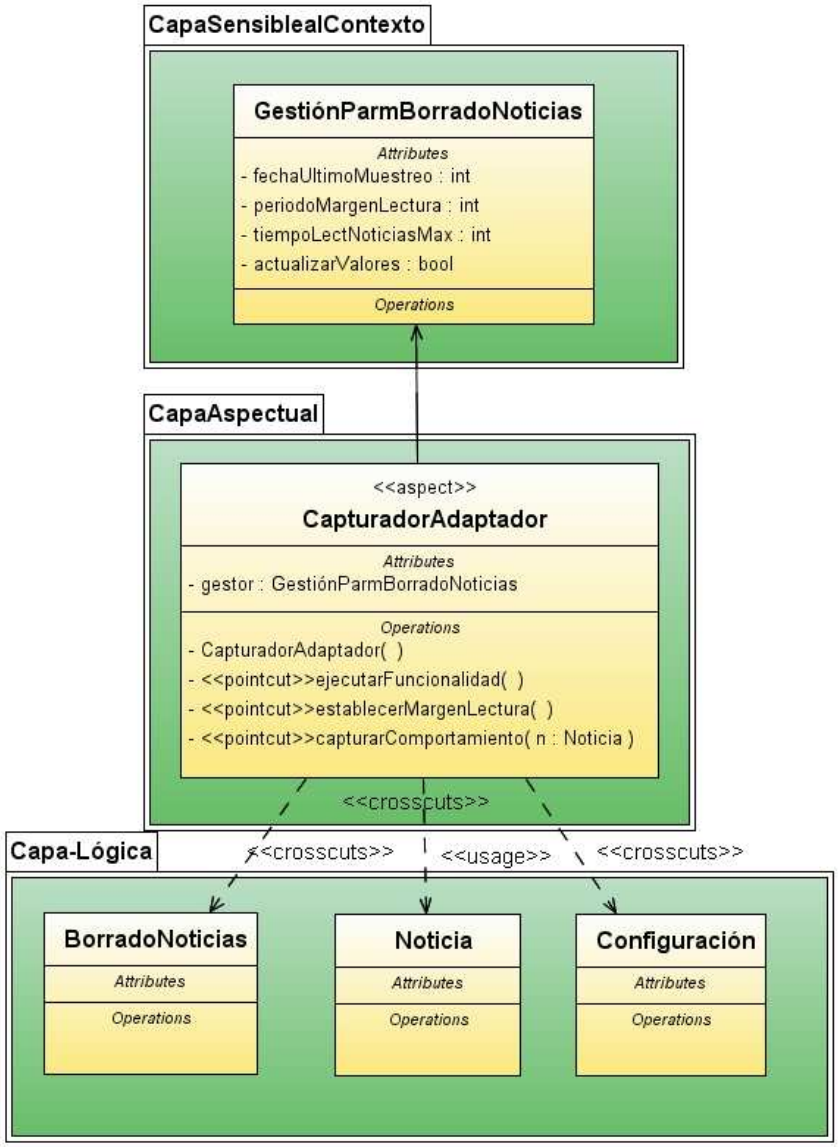


Figura 7.7: *Motor de Plasticidad Implícita* para objetivo de personalización C (*LectorNoticias*).

Factor ambiental a tratar. El nivel de brillo de la pantalla en función de la intensidad de luz ambiental.

Frecuencia y procedimiento de consulta del sensor. Dadas las peculiaridades del caso de estudio tratado aquí, el usuario puede requerir fijar su mirada en la pantalla durante largos periodos de tiempo sin que forzosamente necesite estar interactuando con el sistema. Justamente, esta circunstancia puede darse mientras lleva a cabo la lectura de

las noticias descargadas.

Es por ello que se estima conveniente mantener activada la autorregulación del brillo de pantalla de forma cíclica y constante, independientemente de que el usuario interactúe o no con el sistema. Para ello se destina un *thread* a actuar en paralelo con el sistema para proceder a la consulta cíclica del sensor de luz. La frecuencia de actuación del *thread* depende de las expectativas previstas, aunque no es necesario que sea demasiado frecuente, tratando con ello de limitar el consumo excesivo de recursos de computación por parte del *thread*.

La combinación de una consulta sistemática con el sensor con una consulta pautada cada vez que el usuario interactúa con el sistema cubre todas las expectativas para satisfacer la confortabilidad del usuario.

Efectividad de las variaciones. El brillo de pantalla tan sólo se regula cuando la variación en la luminosidad exterior percibida -o simulada- es significativa.

Otras particularidades. Igualmente, esta componente proporciona la utilidad de registrar los valores que han provocado una alteración del brillo de la pantalla, a efectos de disponer de un histórico.

7.2.3.2.2. Componente de regulación automática del brillo de pantalla

A continuación se detallan los aspectos más importantes del diseño de esta componente.

Clases de la capa lógica involucradas

Al igual que en el sistema "ControladoraObrasAumentada", las clases involucradas en el tratamiento de la luminosidad exterior son todas las pantallas que integran la aplicación base.

También es interceptada la clase principal con el propósito de enviar el histórico de la sesión al *servidor de plasticidad* justo antes de finalizar la sesión.

Capa sensible al contexto

La *capa sensible al contexto* en este caso contempla toda la funcionalidad asociada a la consulta de la luminosidad ambiental y consecuente regulación del brillo de la pantalla, al igual que en el sistema "ControladoraObrasAumentada" (véanse los detalles de la misma en la *sección 7.1.3.2.2.*). La única diferencia en relación a la funcionalidad y al diseño de clases es la intervención adicional de una clase para encapsular el *thread* responsable de las consultas cíclicas al sensor de luz externa, tal y como se ha planteado. Se trata de la clase

ThreadSimple, que cada cierto periodo de tiempo activa la comunicación con el sensor de luz.

Capa aspectual

Al igual que en el sistema "ControladoraObrasAumentada", tan sólo es necesario contar con el *aspecto ControlLuminosidad*, encargado de regular el brillo de la pantalla ante una variación suficientemente significativa. Se trata de la unidad de programa que dispara las consultas a realizar sobre el sensor habilitado cada vez que el usuario genera una interrupción como consecuencia de interactuar con la aplicación (*punto de corte ActivarConsultaSensor*), de gobernar la construcción del histórico de muestras obtenidas durante la sesión (*punto de corte cerrarHistorico*), así como de decidir en qué casos es apropiado aplicar un ajuste en el brillo de la pantalla (*punto de corte ControlarCambios*).

Por último, y a diferencia del caso de estudio *'Controlador de Obras'*, el *aspecto ControlLuminosidad* también se encarga de lanzar a ejecución el *thread* (*punto de corte lanzarThread*).

Representación del MPI para esta componente

En la figura 7.8 (pág. siguiente) se muestra el diagrama de clases que representa el MPI resultante.

7.3. Directrices en la construcción de un Motor de Plasticidad Implícita

Una vez presentados los casos de estudio trabajados, y habiendo mostrado el diseño aplicado para cada uno de los componentes del contexto, el siguiente objetivo a abordar es el de describir las directrices y pautas seguidas en la construcción de cada uno de los componentes para ambos sistemas, como resultado del análisis de los casos de estudio. El propósito es el de identificar esquemas de solución genéricos.

7.3.1. Pautas observadas de los Casos de Estudio

A continuación se presentan los distintos aspectos observados en relación a las dependencias entre capas, así como la composición y propósito de cada una de ellas.

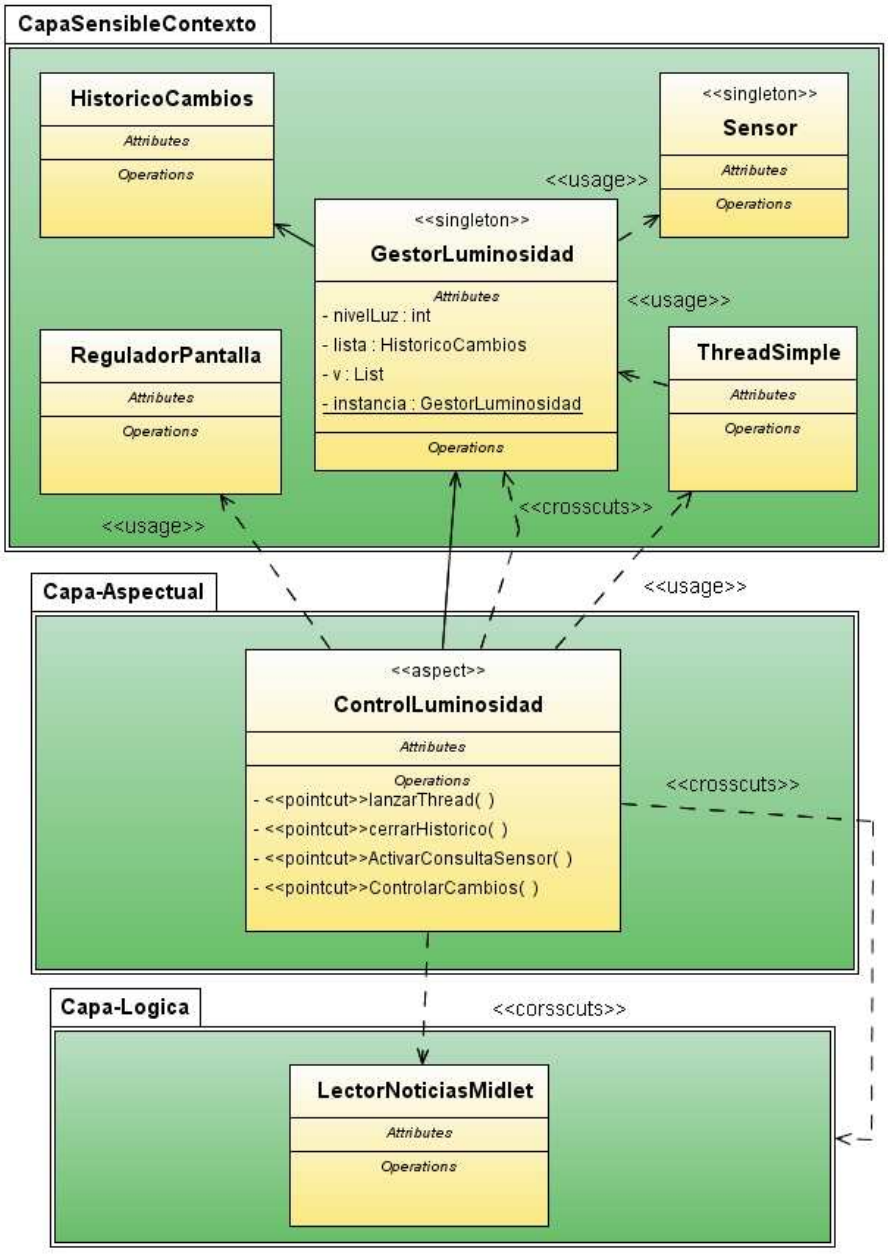


Figura 7.8: *Motor de Plasticidad Implícita* para regular brillo pantalla (*LectorNoticias*).

7.3.1.1. Dependencias

La gran mayoría de dependencias entre capas corresponden al acoplamiento ineludible entre la *capa aspectual* y las otras dos, esto es, las dependencias entre los *aspectos* y las clases introducidas en la *capa sensible al contexto*, así como entre aquéllos y las clases

ya existentes de la *capa lógica*. Este acoplamiento, concretamente el de la *capa aspectual* con la *capa lógica*, que es el más delicado de cara al diseño de un framework genérico, es inevitable para conseguir la meta planteada: reflejar el estado del *modelo contextual* en el sistema base. De lo que se trata es de localizar y modularizar esas dependencias con el sistema lo máximo posible y, si es posible, evitar que el desarrollador de *aspectos* deba tener un conocimiento profundo de las aplicaciones.

El *"principio de la inconsciencia"* (véase *Capítulo 6*; *sección 6.2.3.3.1.*) que caracteriza la POA hace posible alcanzar el objetivo propuesto sin alterar la estructura y código del sistema original, y sin que éste sea consciente de la existencia de las otras dos capas. La ausencia de dependencias de la *capa lógica* hacia las otras dos demuestra el cumplimiento de este principio. Además, garantiza el requisito de transparencia planteado en el *Capítulo 6* (véase *Capítulo 6*; *sección 6.2.1.2.2*).

A continuación se analizan todas las dependencias entre capas.

7.3.1.1.1. Dependencias Capa sensible al contexto – Capa lógica

Ocasionalmente se producen dependencias entre la *capa sensible al contexto* y la *capa lógica* que corresponden a la relación entre las clases que representan el *modelo contextual* y las *entidades objetivo*. Este acoplamiento, que resulta indispensable para ciertos objetivos de personalización al usuario, no se da, sin embargo, en el tratamiento de otro tipo de *restricciones de tiempo real*. En concreto, tan sólo se observa este tipo de dependencias en los objetivos de personalización enfocados a la pantalla. Estas dependencias no violan el requisito de transparencia, dado que la aplicación original tampoco es consciente de la existencia de la *capa sensible al contexto*, ni tampoco sufre ningún tipo de alteración. No obstante, sí suponen un problema de genericidad al manejarse en la *capa sensible al contexto* una representación de la *entidad objetivo* específica de cada sistema. En el *Capítulo 8* se expone la estrategia utilizada para atacar esta problemática.

El caso contrario es el de la componente de adaptación al entorno o recurso hardware, donde se observa una total independencia entre estas capas. La explicación de este comportamiento está en que la fuente de información en una componente de adaptación al entorno no procede de la observación de la ejecución o de la evolución de la actividad, sino que procede del mundo exterior o bien, en el caso de las restricciones en recursos hardware, de la exploración del propio estado físico del equipo. Como consecuencia, existe una total independencia de la *capa sensible al contexto* con respecto a la *capa lógica*. En este caso se consigue delegar totalmente el acoplamiento entre dichas capas en la *capa aspectual*. Por supuesto, esto repercute claramente en favor de la ortogonalidad y la reutilización (metas marcadas inicialmente) de la *capa sensible al contexto* con respecto al sistema base.

Además, aplicando los principios de un buen diseño software es posible incluso aislar las variaciones propias de los distintos factores del entorno o restricciones hardware. En otras palabras, un mismo diseño es válido en independencia del método seguido tanto en la adaptación a aplicar como en la manera como comunicarse con el sensor habilitado en cada caso. Este es el motivo por el que ambos objetivos siguen un mismo diseño. La representación del *modelo contextual* actúa como fuente de alimentación en relación a un cierto parámetro contextual de la que ir consumiendo los sucesivos valores conforme van siendo capturados y proporcionados o bien por el sensor oportunamente habilitado, o bien por parte de la fuente de alimentación previamente dispuesta, si se recurre a una simulación.

7.3.1.1.2. Dependencias Capa aspectual – Capa lógica

Como ya se ha mencionado, las dependencias entre la *capa aspectual* y la *capa lógica* son inevitables. Tal y como se describe en el *Capítulo 8*, con objeto de eludir y separar al máximo el código común del código específico para un sistema determinado se construyen jerarquías de *aspectos*, de manera que los *aspectos* base concentran las partes comunes y directamente reutilizables, dejando las partes específicas para los *sub-aspectos*, a ser especializados para cada sistema.

Como se ha podido observar, la inclusión de declaraciones inter-tipo es necesaria en la mayoría de componentes. Por supuesto, este factor introduce dependencias. La estrategia seguida ha sido la de encapsular todas esas declaraciones en un *aspecto* específico para ello, con el propósito de aislar también este tipo de dependencias. Este *aspecto* es el denominado *IncrementadordeOperaciones*.

Las clases de la capa lógica involucradas por la *capa aspectual* responden también a un patrón común. Suelen verse afectadas en todos los componentes la clase principal y la clase *entidad objetivo*. Adicionalmente, en los componentes de personalización de una pantalla hay que añadir las propias pantallas involucradas. En la de personalización de una funcionalidad de la aplicación base se ve involucrada la clase que encapsula dicha funcionalidad. En lo que respecta a la componente de apoyo al trabajo en grupo, las *entidades objetivo* afectadas son aquellas directamente implicadas en los aspectos de colaboración, que en el caso tratado han sido las clases *Tarea* y *Material*.

Finalmente, en la componente de adaptación al entorno o recurso hardware, las clases de la *capa lógica* involucradas pueden ser muy diversas dependiendo de la *restricción de tiempo real* objeto de tratamiento. En general, cuando de lo que se trata es de regular un determinado parámetro de la IU es necesario interceptar las clases directamente relacionadas con la IU. No obstante, en ocasiones el tratamiento de determinados factores

ambientales, como por ejemplo el nivel de sonido o la localización del usuario puede ser relevante únicamente en la ejecución de ciertas partes muy puntuales del sistema. Por ejemplo, en el momento de inicio de una cierta reproducción, o en el momento en que el usuario consulta su ubicación en un mapa virtual de la zona.

En el tratamiento de las restricciones en recursos hardware, el que por ejemplo el nivel de batería, la conexión de red o la memoria disponible sean críticas y se requiera un determinado tratamiento, generalmente afecta en la ejecución de funcionalidades concretas del sistema. Las clases implicadas son en estos casos las que se encargan específicamente de esas funcionalidades.

7.3.1.1.3. Dependencias Capa aspectual – Capa sensible al contexto

En todos los casos estudiados el *aspecto* principal o *aspectos* principales se componen de un objeto de la clase principal de la *capa sensible al contexto*, la cual refleja el estado del *modelo contextual*. El acoplamiento establecido entre estas dos capas entra dentro de lo previsto, no suponiendo ningún tipo de inconveniente de cara a preservar las propiedades de transparencia y reutilización planteadas inicialmente. De hecho, la funcionalidad de las *capas sensible al contexto* y *aspectual* se complementa mutuamente.

De nuevo, las dependencias son siempre de la *capa aspectual* hacia la *capa sensible al contexto*, de manera que ésta última no es tampoco consciente de la existencia de la *capa aspectual*, resultándole transparente la manera como se hace uso de la funcionalidad ofrecida. Por supuesto, de no existir la *capa aspectual*, la funcionalidad de la *capa sensible al contexto* debería ser incorporada directamente en la *capa lógica*, lo que supondría incurrir en la problemática de enmarañamiento y dispersión de código propios de la aplicación del paradigma *orientado a objetos* en este tipo de sistemas, tal y como se apunta en el *Capítulo 6*, violando las propiedades de transparencia, ortogonalidad y reutilización perseguidos. De hecho, estos problemas se ponen en evidencia en los experimentos realizados, los cuales se describen a la *sección 7.4*.

7.3.1.2. Composición y propósito de cada capa

A continuación se presenta en términos generales la composición y propósito de cada capa. En el *Apéndice B* se proporcionan los detalles de diseño e implementación expresados de manera genérica, esto es, sustituyendo las referencias específicas del sistema por referencias genéricas. Así, para hacer referencia de forma genérica a la clase que representa la *entidad objetivo* involucrada en cada sistema se utiliza el identificador de clase EntdObj. Además, se generalizan las particularidades identificadas en la descripción de los casos

de estudio con el propósito de obtener un diseño y una implementación suficientemente genéricos para su validez ante diversas situaciones y necesidades. Se trata, por tanto, de los esquemas de solución de cada uno de los componentes.

7.3.1.2.1. Capa aspectual

La *capa aspectual*, como capa intermedia, absorbe la mayoría de las dependencias, desempeñando el papel de conductora y mediadora entre las otras dos capas, tal y como se pretendía. Es la única que es consciente de la existencia de las otras dos, estableciendo un enlace transparente entre ellas, tal y como se propone en el *Capítulo 6*. En definitiva, gobierna la intervención de la funcionalidad de la *capa sensible al contexto*, interceptando piezas de código clave de la *capa lógica*. La inyección de código se realiza en base a dos funciones distintas:

1. capturar información acerca de: (1) el patrón de actuación del usuario en el uso de la IU o en la realización de ciertas acciones; y/o (2) el estado de los objetos correspondientes a las *entidades objetivo*, a fin de construir y actualizar el *modelo contextual*.

   La captura de la información necesaria se realiza interceptando puntos clave de la ejecución del sistema, estableciendo los oportunos *puntos de corte*.

   Cabe destacar que en el tratamiento de *restricciones de tiempo real* relacionadas con el entorno, este tipo de *puntos de corte* no se presentan, puesto que la captura de información se realiza a través del uso de sensores encargados de percibir el mundo exterior, no siendo necesario observar en estos casos la evolución que va siguiendo la ejecución.

2. plasmar la información recopilada en el *modelo contextual* en el curso de la ejecución, a fin de conseguir el objetivo de adaptación propuesto.

Si una aplicación requiere más de un objetivo de adaptación se reúnen todos los *aspectos* en una misma *capa aspectual*. La propiedad de ortogonalidad permite que la aplicación pueda evolucionar fácilmente pudiendo añadir y quitar las funcionalidades extra según las necesidades que deba afrontar a cada momento. La problemática relacionada con las *interacciones entre aspectos* se resuelve a través del mecanismo de *precedencia entre aspectos*, tal y como se explica en el *Capítulo 6* (véase *Capítulo 6*; *sección 6.2.3.3.4.*).

A continuación se exponen los propósitos de esta capa para las distintas componentes.

Componente de personalización

Dadas las peculiaridades que caracterizan cualquier objetivo de personalización, donde los *inputs* proceden del uso que se hace de la aplicación o de la IU por parte del usuario, debe incluirse algún mecanismo de monitorización a efectos de capturar el patrón de actuación del usuario.

Personalización de una pantalla

El tratamiento llevado a cabo por el *aspecto CapturadorAdaptador* incluye las siguientes operaciones: (1) materialización de la tabla al inicio del programa, a fin de soportarla en memoria durante toda la ejecución; (2) serialización de la tabla al finalizar el programa, a fin de garantizar su persistencia; (3) actualización de la tabla con la combinación de parámetros introducida (objetivo de personalización *'valores por defecto en un formulario'*) o la opción de la lista seleccionada (objetivo de personalización *'ordenación de una lista'*), cada vez que el usuario se dispone a procesar el formulario objeto de personalización; y (4) utilización de la tabla de *hash* para obtener o bien el *'valor de moda'*, o bien la ordenación de las opciones seleccionadas por el usuario, a fin de ser plasmadas en la IU.

Con objeto de garantizar la inyección de todas estas acciones oportunamente en la ejecución del sistema base, se destina un *punto de corte* distinto para cada uno de los tratamientos mencionados sobre la tabla de *hash*. La tercera responsabilidad -actualización de la tabla-, tiene claramente una función de captura del patrón de actuación del usuario, indispensable para llevar a cabo la personalización, a fin de recopilar e inferir sus preferencias.

Por supuesto, todo este esfuerzo por mantener la tabla actualizada con todas las ocurrencias no es otro que el de adaptar la propia pantalla, acción que se realiza en el momento de inicializar la pantalla en cuestión. Para ello se requiere un cuarto *punto de corte*, el cual tiene asignada una función de adaptación, dado que una vez haber recopilado el historial de interacción del usuario y haber aplicado el tratamiento oportuno, en función del objetivo de personalización y de la heurística aplicada, se puede proceder a ejecutar la adaptación pertinente en la pantalla objeto de personalización.

Cabe mencionar aquí que para el objetivo de personalización *'ordenación de una lista'* no sólo es necesario interceptar el método constructor para personalizar la lista propiamente dicha, sino que además es necesario personalizar la propia acción de seleccionar las distintas opciones, dado que responden a una lista personalizada de la cual únicamente el *aspecto CapturadorAdaptador* tiene el control. En definitiva, debe ser el propio *aspecto* el que se encargue de procesar la selección de cada opción.

Personalización de una funcionalidad base

El aspecto CapturadorAdaptador se compone de un objeto de la clase gestora perteneciente a la *capa sensible al contexto* –denominada *GestiónParmFuncionalidadExistente* en el caso de estudio-, a fin de darle el tratamiento oportuno.

El tratamiento a aplicar incluye las siguientes operaciones: (1) materialización de los valores implicados en la clase *GestiónParmFuncionalidadExistente* al inicio de la ejecución, a fin de ser soportados en memoria durante toda la ejecución; (2) actualización del parámetro objeto de personalización, una vez transcurrido el periodo de muestreo establecido; y (3) serialización de los valores soportados por la clase *GestiónParmFuncionalidadExistente*, a fin de garantizar su persistencia.

Componente de apoyo al trabajo en grupo

En general, una componente de apoyo al trabajo en grupo completa debe ir provista de las tres metas propias de una actividad colaborativa: comunicación, coordinación y colaboración. No obstante, podría darse el caso de que para un sistema en particular no fuera necesario complementar la funcionalidad del sistema en uno de los tres aspectos. El hecho de que se propongan tres unidades de modularización *aspecto* distintas para cada uno de ellos evita problemas y otorga la máxima ortogonalidad, al tratarse de manera independiente cada una de las metas.

La complementación de la operativa básica del sistema se lleva a cabo a través de una combinación de las acciones siguientes, a llevar a cabo por estos tres *aspectos*: (1) incrementar con aspectos colaborativos la funcionalidad básica; (2) comunicarse con el servidor para hacerle partícipe de ciertas circunstancias o incidencias de relevancia para el desarrollo de la actividad grupal; y (3) actualizar la *consciencia de grupo particular*.

Componente de adaptación al entorno o recurso hardware

De acuerdo a las observaciones extraídas de los casos de estudio tratados, es suficiente con una única unidad de programa *aspecto* por cada *restricción de tiempo real*, la cual es responsable de mecanizar la comunicación con el sensor a fin de disponer de los sucesivos valores capturados con una cierta regularidad y ante unas determinadas circunstancias. Este *aspecto* intercepta aquellos puntos de la ejecución del programa donde la aplicación de un determinado tratamiento o ajuste de la pantalla, si procede, es recomendable –en ocasiones necesaria- para mejorar la confortabilidad del usuario, o bien para completar una cierta funcionalidad con éxito.

Si lo que se pretende es aplicar pequeños ajustes a la IU, como es el caso del brillo de pantalla, es suficiente con interceptar las clases relativas a la IU, a fin de conocer cuándo el usuario está interactuando. Si lo que se pretende es adaptar la realización de

cierta funcionalidad a las restricciones en recursos hardware monitorizadas, las clases a interceptar son específicamente las responsables de dicha funcionalidad.

7.3.1.2.2. Capa sensible al contexto

Tal y como se introduce en el *Capítulo 6* (véase *Capítulo 6*; *sección 6.3.1.2.*), esta capa está enfocada en la representación, construcción y mantenimiento de toda la información necesaria para representar aquellos atributos del contexto (las denominadas *restricciones de tiempo real* –véase *Capítulo 2*; *sección 2.1.3.*-) en los que se está interesado y ofrecer los medios necesarios para obtener de él, una vez capturado y tratado, las pertinentes deducciones y observaciones en base a ciertas heurísticas o criterios.

Entre las responsabilidades desempeñadas en esta capa se encuentran las siguientes: (1) mantener la información en la memoria del dispositivo –en la mayoría de ocasiones también en la memoria persistente-; (2) procesarla y prepararla para la adaptación; y (3) ofrecer los medios necesarios para su consulta. Además, dependiendo de la procedencia de los *inputs* que influyen en la adaptación –del exterior o de la observación de la propia ejecución-, la responsabilidad de capturar y actualizar la información relativa a la *restricciones de tiempo real* recae o bien en la *capa aspectual* o bien en la *capa sensible al contexto*.

A continuación se sintetizan los propósitos de las distintas componentes, por grupos de objetivos.

Objetivos de personalización

El tratamiento de la personalización implica una monitorización del patrón de actuación y preferencias del usuario, a capturar por parte de la *capa aspectual*, que es la que también se encarga de actualizarla utilizando las instancias de las clases de la *capa sensible al contexto* de que se componen los *aspectos*. Los *inputs* proceden del uso que se hace de la aplicación y de la IU por parte del usuario. En estos casos la *capa sensible al contexto* ofrece las clases de soporte para mantener esa información en memoria y, si procede, también en la memoria persistente.

Objetivo de apoyo al trabajo en grupo

Los objetivos planteados en una componente de apoyo al trabajo en grupo implican la monitorización de ciertas incidencias, circunstancias o acciones que implican a otros miembros del grupo. Lo *inputs* proceden del contexto y circunstancias que rodean la actividad colaborativa, y que sólo son detectables monitorizando el cumplimiento de ciertas condiciones en el programa base, o bien la realización de ciertas acciones puntuales por

parte de los usuarios. Por lo tanto, también en este tipo de componente se requiere observar el desarrollo de la actividad colaborativa. Es la *capa aspectual* la que se encarga de llevar a cabo esta monitorización, así como de proceder a la actualización del *modelo contextual*, a fin de construir la denominada *consciencia de grupo particular*.

Objetivo de adaptación al entorno o recurso hardware

En este caso no es la capa *aspectual* la encargada de realizar la monitorización, sino que es la propia capa *sensible al contexto* la responsable de obtener esos *inputs*. No obstante, la responsable de procesar los cambios percibidos es, de nuevo, la *capa aspectual*.

Como la información relativa a estas *restricciones de tiempo real* no proviene de la ejecución de la aplicación o desarrollo de la actividad, sino del mundo exterior o del estado del propio equipo, se requieren los sensores apropiados y las APIs necesarias para su captura y tratamiento a nivel de programa.

7.4. Parte experimental

Con la intención de evaluar el impacto del uso de *aspectos* para incorporar funcionalidad *sensible al contexto* en aplicaciones móviles, se han implementado distintas versiones de una misma aplicación, en las que se integra esta funcionalidad extra utilizando POA en unos casos y utilizando estrictamente POO en otros, partiendo de un mismo código inicial. La aplicación con la que se ha experimentado es el *'Lector de Noticias'* presentado anteriormente.

A continuación se presenta el método y las métricas utilizadas en el análisis comparativo, los resultados obtenidos y una discusión final conclusiva.

7.4.1. Método y métricas utilizadas en el análisis comparativo

Partiendo de la versión original de la aplicación "LectorNoticias" descrita en la *sección 7.2.1.*, se han desarrollado distintas versiones que presentan un mismo comportamiento *sensible al contexto*. Para demostrar que efectivamente las versiones *orientadas a objetos* presentan los dos síntomas de dispersión y enmarañamiento de código -cuya aparición justifica recurrir a una técnica de *separación de conceptos*-, se han contabilizado las líneas de código destinadas a la nueva funcionalidad, diferenciando la parte que ha podido ser encapsulada en clases de la que se presenta enmarañando la funcionalidad núcleo de la aplicación.

Se presentan dos experimentos para tratar por separado la adaptación al entorno y la personalización al usuario. En este último caso se integran las tres componentes descritas anteriormente para los objetivos de personalización A, B y C (véanse *secciones 7.2.3.1.1.*, *7.2.3.1.3.* y *7.2.3.1.5.*). Cabe señalar que el número de versiones implementadas es distinto en ambos experimentos. Eso es debido a que una vez haber analizado el impacto de utilizar o no el patrón *Singleton* [GHJV95] (es el único patrón que es susceptible de ser aplicado en todos los componentes desarrollados) tanto en las versiones *aspectuales* como en las *orientadas a objetos* en el primer experimento, en el segundo se opta por desarrollar exclusivamente la opción más favorable para ambos casos.

Las métricas utilizadas para el análisis comparativo entre versiones han sido las mismas en ambos experimentos, tomándose los valores pertinentes para todas y cada una de las versiones desarrolladas. Son las siguientes:

1. *Extensión del código fuente.* Para estudiar la repercusión en la extensión del código fuente se han tomado los siguientes valores:
   a) el número total de líneas de código fuente
   b) el número total de líneas de código que supone la nueva funcionalidad. En relación a este último valor se ha prestando un interés especial en distinguir entre
      1) el código que ha podido ser encapsulado en clases y
      2) el código que debe ser insertado en la funcionalidad núcleo para invocar la nueva funcionalidad. Es precisamente este valor el que nos permite conocer la proporción de enmarañamiento de código, al tratarse de instrucciones que se insertan y entremezclan con el código original. Precisamente, las versiones *aspectuales* presentan todo el código correspondiente a la nueva funcionalidad encapsulado y aislado de la aplicación base.
      Con objeto de constatar el nivel de enmarañamiento de código al incorporar la nueva funcionalidad, se calcula
      3) el porcentaje de código enmarañado con respecto al total de líneas de código que supone introducir la nueva funcionalidad. Otro parámetro interesante y que también se recoge es
      4) el número de clases de la aplicación base afectadas por la nueva funcionalidad, esto es, en las que se entremezcla código.
2. *Tamaño del bytecode*[6]. Para estudiar la repercusión en el tamaño de la aplicación final se han tomado los siguientes valores:

[6] código una vez compilado, también denominado código intermedio.

a) el tamaño del *bytecode* correspondiente exclusivamente al código fuente

b) el tamaño del *bytecode* total (el fuente junto con todas las librerías y recursos necesarios para ser desplegado). El tamaño de las librerías y recursos que intervienen también se proporciona. Cabe señalar que una diferencia considerable entre las versiones *orientadas a objetos* y *aspectuales* es la inclusión o no del runtime *aspectjrt*.

c) el tamaño del *bytecode* comprimido, esto es, el del archivo *jar* a ser desplegado en el dispositivo.

Todos estos valores se ofrecen en Kbytes. Con objeto de facilitar la comparativa se ofrece también el tamaño del código fuente antes de ser compilado. Esto permite contrastar el impacto de aplicar el paso de *weaving* (compilación más paso de recomposición de los aspectos con la aplicación base) en las versiones *aspectuales*, con respecto al de aplicar la compilación en las versiones *orientadas a objetos*. Este valor corresponde a la primera entrada de la tabla, a fin de dar una idea de la evolución que sigue el proceso de generación del archivo *jar*.

Además, en las comparativas de ambos experimentos se muestra también esos mismos datos con respecto a la aplicación original. Estos datos proporcionan una referencia a efectos de observar el efecto que supone integrar la nueva funcionalidad.

Como consecuencia del enmarañamiento de código que se pone de manifiesto, la legibilidad, facilidad de mantenimiento y trazabilidad de código en las versiones *orientadas a objetos* se ve visiblemente reducida, mientras que las versiones *aspectuales* no se produce ningún tipo de impacto en el código fuente de partida. Aunque este tipo de deducciones y observaciones son subjetivas, y por tanto de difícil cuantificación, tras haber implementado las distintas versiones y haber analizado todos los resultados se describen las observaciones extraídas acerca del nivel de acoplamiento y dependencia entre clases, así como de las consecuentes repercusiones en la evolución del código. No obstante, los diseños aplicados en las versiones *aspectuales*, plasmados en las figuras anteriores, dan una idea aproximada del nivel de acoplamiento y precisan qué componentes están implicadas en esas dependencias.

En las versiones *orientadas a objetos*, sin embargo, al no disponer de una capa intermedia (la *capa aspectual*), las clases de la funcionalidad núcleo mantienen numerosas referencias a las clases del paquete de luminosidad/personalización, implicando a un número considerable de clases (valor de la fila número 6 en la tabla 7.1). En definitiva, el acoplamiento entre el código base y la nueva funcionalidad es importante.

7.4.2. Experimentación con la componente de regulación del brillo de pantalla

En esta sección se describe el primer experimento llevado a cabo integrando en la aplicación "LectorNoticias" la componente de adaptación al entorno descrita en la *sección 7.2.3.2.* para regular automáticamente el brillo de pantalla.

A continuación se detallan las distintas versiones desarrolladas para pasar seguidamente a presentar todos los valores obtenidos para cada una de las métricas consideradas.

7.4.2.1. Descripción de las versiones desarrolladas

En este experimento se han desarrollado cuatro versiones distintas de la aplicación, todas ellas presentando un comportamiento idéntico. Son las siguientes:

1. la implementación propuesta anteriormente en la *sección 7.2.3.2.2.*, en la que la forma de estructurar las distintas componentes se ajusta a la arquitectura de software descrita en el *Capítulo 6*. Esta versión se ajusta al cien por cien con el diseño presentado en esta componente (véase figura 7.8), donde la clase *GestorLuminosidad* y *Sensor* se presentan como un *Singleton*. En adelante se hace referencia a esta versión como versión *Aspectual*.

2. la misma implementación anterior con la única diferencia de que la clase *GestorLuminosidad* no está implementada como un *Singleton*. La intención es observar el impacto de utilizarlo tanto en la versión con *aspectos* como en la versión *orientada a objetos*. En adelante se hace referencia a esta versión como versión *Aspectual*-{*Singl*}.

3. implementación en la que la nueva funcionalidad ha sido integrada utilizando estrictamente *programación orientada a objetos* haciendo uso de los patrones GoF *Singleton* y *Observador* (éste último utilizado para reflejar en la IU los cambios detectados en el nivel de luminosidad ambiental percibidos a través del sensor de luz) [Gamma el al., 95]. En adelante se hace referencia a esta versión como versión *OObjetos*.

4. la misma implementación anterior con la única diferencia de que la clase *GestorLuminosidad* no está implementada como un *Singleton*, a fin de disponer de dos versiones lo más equivalentes posible con las correspondientes versiones *aspectuales*, tratando de precisar al máximo en la comparativa realizada. En adelante se hace referencia a esta versión como versión *OObjetos*-{*Singl*}.

Con respecto a las versiones *aspectuales*, una opción hubiera podido ser utilizar una implementación *orientada a aspectos* para el patrón observador, tal y como se propone en [HK02], aunque en la versión *aspectual* descrita en la *sección 7.2.3.2.2.* el *aspecto ControlLuminosidad*, a través del *punto de corte ControlarCambios*[7] se comporta como un *observador*, procediendo a la regulación del brillo de la pantalla conforme detecta cambios significativos.

7.4.2.2. Resultados experimentales

Tal y como se expone a continuación, las versiones *aspectuales* consiguen reducir el número de líneas de código para implementar la funcionalidad *sensible al contexto*, la cual queda completamente encapsulada en las unidades *aspecto*. No obstante, el tamaño del *bytecode*, y por tanto del archivo desplegable, resulta considerablemente superior en las versiones *aspectuales* en comparación con las *orientadas a objetos*. El motivo de este comportamiento se detalla a continuación.

7.4.2.2.1. Extensión del código fuente

Efectivamente, tal y como se esperaba, en las versiones *orientadas a objetos*, a pesar de que el código de la nueva funcionalidad está separado en clases, la líneas de código encargadas de resolver las sucesivas peticiones de control y gestión de la luminosidad aparecen irremediablemente diseminadas a lo largo de la aplicación enmarañando la funcionalidad núcleo que, en consecuencia, pierde cohesión. Se trata de código duplicado y redundante –problema de dispersión de código- correspondiente a instrucciones de programa que en las versiones *aspectuales* tan sólo es necesario escribir una sola vez.

Tal y como cabría esperar, el efecto del enmarañamiento de código es más relevante en la versión *OObjetos-{Singl}* que en la versión *OObjetos*. En efecto, la utilización del patrón *Singleton* reduce significativamente las dependencias entre clases puesto que éstas pueden acceder directamente a la única instancia de la clase (visibilidad global), sin necesidad de pasarse mutuamente la instancia de la clase *GestorLuminosidad* a través del uso de parámetros o métodos extra, reduciendo por tanto el enmarañamiento de código. Comparando el número de líneas de código disperso (línea número cuatro de la tabla 7.1) en una y otra versión (*OObjetos-{Singl}* y *OObjetos*) se puede observar el impacto de su aplicación.

[7] Este *punto de corte* recurre al designador *set* –véase *Apéndice A*- aplicado sobre el valor de la luminosidad externa actual registrada en el gestor (clase *GestorLuminosidad*).

La tabla 7.1 recoge todos los resultados obtenidos en relación a las métricas relacionadas con la extensión del código fuente, las cuales se detallan en el apartado (a) en la *sección 7.4.1*. Las sucesivas líneas corresponden a los seis ítems que allí se describen, siguiendo ese mismo orden.

	Versión original	**(a) Aspectual**	**(b) Aspectual-{Singl}**	**(c) OObjetos**	**(d) OObjetos-{Singl}**
Líneas totales código	5429	5753	5746	5783	5826
Código lumin. total	--	324	317	354	397
Líneas lumin. encapsul.	--	324	317	279	284
Líneas lumin. dispersas	--	0	0	75	113
% código disperso	--	0	0	21,1	28,5
N^o clases afectadas	--	0	0	19	20

Cuadro 7.1: Comparativa en la extensión del código fuente y efecto de enmarañamiento entre las distintas versiones en la componente de regulación del brillo.

Así como en las versiones *orientadas a objetos* resulta conveniente utilizar el patrón *Singleton*, por la reducción tanto en las líneas de código como en el acoplamiento, en las versiones *aspectuales* el resultado obtenido es distinto. Como se observa en la tabla 7.1, el número de líneas de código final es menor cuando no se aplica el *Singleton*. La explicación está en el hecho de que en la versión *aspectual* las únicas unidades de programa que conocen y utilizan la clase *GestionLuminosidad* son el *aspecto ControlLuminosidad* y el la clase *ThreadSimple*, y por lo tanto es suficiente con que el *aspecto* le pase a ésta la única referencia a aquélla clase una sola vez. El uso de este patrón en las versiones *aspectuales* es irrelevante. Es más, el incremento en las líneas de código empleadas en la versión *Aspectual* –con patrón *Singleton*- corresponde a las líneas de código que supone implementar este patrón.

Esta es otra observación que pone de manifiesto el enmarañamiento de código y dependencias introducidas en las versiones *orientadas a objetos*, puesto que se hace patente la existencia de varias clases que tienen visibilidad sobre la clase *GestionLuminosidad* y, por tanto, invocan sus métodos, mientras que en las versiones *aspectuales* las clases de

la *capa lógica* permanecen al margen de la nueva funcionalidad. Esta observación confirma la *Hipótesis 3* planteada en el *Capítulo 1* (véase *Capítulo 1*; *sección 1.3.*), en la que se plantea el hecho de que el tratamiento de los diversos factores contextuales genera *conceptos transversales*.

Por otro lado, en las versiones *aspectuales* se consigue concentrar todo el código correspondiente a la nueva funcionalidad en unidades de programa separadas, de manera que la funcionalidad núcleo permanece intacta, tal y como indican las tres últimas líneas de la tabla 7.1. Esto confirma otra de las hipótesis planteadas en el *Capítulo 1* (véase *Hipótesis 4*; *Capítulo 1- sección 1.3.*).

Cabe recordar que, tal y como se menciona en el *Capítulo 6*, la funcionalidad extra introducida se reparte entre la *capa aspectual* y la *capa sensible al contexto*, a efectos de optimizar la reutilización de código y facilitar la legibilidad de los *aspectos*. Por lo tanto, el número de líneas de código de la tercera línea de la tabla está repartido entre clases y *aspectos*.

A continuación se presentan los resultados correspondientes a la segunda métrica utilizada.

7.4.2.2.2. Tamaño del bytecode

Con respecto a la segunda métrica estudiada: el tamaño del *bytecode*, los resultados no son favorables a las versiones *aspectuales*. La tabla 7.2 recoge todos los valores obtenidos relacionados con esta métrica, todos ellos expresados en Kbytes. La primera línea muestra el tamaño del código fuente previo al paso de compilación/*weaving*, con la intención de poner de manifiesto cómo impacta la aplicación de este paso en el tamaño del *bytecode* resultante en ambos tipos de versiones (*aspectual* y *orientada a objetos*). La segunda línea muestra el tamaño del código fuente una vez compilado (compilación y recomposición en el caso de las versiones *aspectuales*). La tercera línea muestra el tamaño de las librerías y recursos necesarios para el despliegue de la aplicación. La cuarta indica el tamaño de la librería runtime de AspectJ, necesaria para su funcionamiento en las versiones *aspectuales*. La quinta corresponde al tamaño total del *bytecode*, y, por último, se muestra el tamaño resultante, una vez empaquetado y listo para ser desplegado en el dispositivo móvil.

Tal y como queda plasmado en la tabla, a pesar de que las versiones *aspectuales* requieren menos líneas de código, el paso de recomposición (*weaving*) resulta considerablemente ineficiente en comparación al proceso de compilación en las versiones *orientadas a objetos*, lo que se contribuye a engrosar el código resultante. De hecho, el tamaño de la versión final a ser entregada (sexta línea de la tabla 7.2) se incrementa en casi un 49 % con respecto a las versiones *orientadas a objetos*. Esto no se debe al proceso propio de

	Versión original	**(a) Aspectual**	**(b) Aspectual-{Singl}**	**(c) OObjetos**	**(d) OObjetos-{Singl}**
Tamaño código fuente	144	152.5	152.3	153.2	154.4
***bytecode* código fuente**	144	205	203	159	161
Recursos y librerias	74.4	74.4	74.4	74.4	74.4
Líbreria *AspectJrt*	--	204	204	--	--
tamaño total *bytecode*	219	483	481	234	236
***bytecode* comprimido**	124	259	258	132	132

Cuadro 7.2: Comparativa en el tamaño del bytecode entre las distintas versiones en la componente de regulación del brillo.

recomposición de los *aspectos* con las clases de la funcionalidad núcleo, sino que es debido a ciertas limitaciones de J2ME respecto a J2SE y al hecho de que no se disponga de una versión de *AspectJ* específica para J2ME, tal y como se detalla a continuación.

En efecto, el *weaver* (compilador y entretejedor) de *AspectJ* instrumenta el código Java para proporcionar ciertas capacidades de reflexión a las clases que no las tienen, lo que provoca la generación e incorporación de clases extra de tamaño considerable, e incluso la necesidad de incrementar el tamaño de las clases existentes. Este tipo de aspectos son ignorados en un proceso de compilación de una aplicación J2ME común porque, efectivamente, estas características no pueden ser aprovechadas por una aplicación J2ME. Precisamente es el paso de preverificación el que se encarga de evitar la incorporación de este tipo de clases específicas de la versión J2SE, preservando las restricciones de J2ME. Sin embargo, para conseguir enlazar la librería *aspectjrt* con la aplicación ha tenido que ser ignorado explícitamente el paso de preverificación propio de J2ME. El código resultante, aunque no cumple dicha preverificación funciona correctamente y trabaja con *aspectos* como si se tratara de una aplicación J2SE. La reflexión es una de las limitaciones de J2ME con respecto a J2SE, limitación que debe ser suplida para conseguir enlazar una librería propia de J2SE (en este caso la librería de *AspectJ*).

Por otro lado, además de la ineficiencia del paso de *weaving* por los motivos expuestos, hay que tener en cuenta que en las versiones *aspectuales* se requiere, además, enlazar la

librería *AspectJrt* para que el código desplegable sea ejecutable en la JVM del dispositivo destino. Esto contribuye a engrosar aún más el tamaño final del *bytecode*, tal y como se muestra en la tabla 7.2. La librería *AspectJrt* por sí sola ya ocupa 204 Kbytes. Al respecto de este problema, ya ha sido anunciado que una nueva versión de *AspectJ* para J2ME estará operativa en breve. Mientras tanto, esta es la única manera de conseguir trabajar con *AspectJ* en aplicaciones J2ME.

No obstante, a pesar del incremento del tamaño final del desplegable, estas versiones de la aplicación se han podido ejecutar con éxito tanto en los emuladores como en los dispositivos destino: teléfonos móviles de segunda generación[8]. Se constata por tanto que, a pesar de los problemas de eficiencia y las dificultades derivadas de no disponer hasta el momento de una versión optimizada de *AspectJ* para J2ME, la meta de obtener una versión *aspectual* operativa asumible por un dispositivo limitado se ha alcanzado con éxito.

7.4.3. Experimentación con la componente de personalización

A continuación se describe el segundo experimento llevado a cabo integrando en la aplicación "LectorNoticias" las tres componentes de personalización descritas en las *secciones 7.2.3.1.1.*, *7.2.3.1.3.* y *7.2.3.1.5.* Se comienza detallando las distintas versiones desarrolladas y posteriormente los resultados obtenidos en cada una de las métricas consideradas.

7.4.3.1. Descripción de las versiones desarrolladas

En este caso se ha implementado una única versión *orientada a objetos* y una única versión *aspectual*. La versión *aspectual* corresponde a la descrita en la *sección 7.2.3.1.*, siguiendo la estructura software presentada en el *Capítulo 6*. La versión *orientada a objetos* presenta el mismo comportamiento recurriendo exclusivamente al uso de clases para integrar la nueva funcionalidad. Como ya se ha mencionado anteriormente, no se ha considerado relevante volver a experimentar el impacto del patrón *Singleton*, y se ha optado por la opción más favorable en ambos casos. En adelante se denomina a ambas versiones *Aspectual* y *OObjetos* respectivamente.

7.4.3.2. Resultados experimentales

En este caso, tal y como se refleja en las tablas, no se consigue reducir el número de líneas de código requeridas para integrar la nueva funcionalidad haciendo uso de *aspectos*. Esto es debido a que en este caso en la versión *orientada a objetos* no es necesario introducir tanto código repetido como en la componente anterior, por lo que el número de clases

[8]Se trata de dispositivos cuyas capacidades son considerablemente inferiores a las de los dispositivos más avanzados que se encuentran hoy en día en el mercado.

afectadas es también inferior que en el experimento anterior. De hecho, la personalización tratada en estas componentes es muy puntual, afectando tan sólo a un formulario en concreto (objetivo de personalización A), una lista en particular (objetivo de personalización B) y la ejecución de una funcionalidad en particular (objetivo de personalización C). Tan sólo se estudia el patrón del usuario al interaccionar con estas clases puntuales.

En consecuencia, este caso no resulta demasiado significativo de la dispersión de código redundante en la versión *orientada a objetos*. Sin embargo, eso no significa que no se produzcan igualmente dependencias entre clases y un acoplamiento importante entre la funcionalidad núcleo y la que es incorporada. El número de líneas de código entremezclado con la aplicación base es incluso superior al requerido para la componente del experimento anterior y el porcentaje no es inferior al de la versión *OObjetos* de aquél (21,4 % frente a un 21,1 % en el primer experimento). A diferencia del experimento anterior, el código introducido para la nueva funcionalidad no es código repetitivo. Por lo tanto, aunque se incurre en problemas de enmarañamiento, la dispersión de código redundante es reducida o nula. De nuevo, en la versión *aspectual*, tanto el enmarañamiento como la dispersión de código son nulos, por lo que la aplicación base no se ve afectada, lo que constituye una de las metas planteadas inicialmente (véase *Capítulo 6*; *sección 6.2.1.2.2.*).

La tabla 7.3 recoge todos los resultados obtenidos en relación a las métricas relacionadas con la extensión del código fuente para las dos versiones desarrolladas. Las entradas de la tabla son las mismas que para el experimento anterior.

	Versión original	**(a) Aspectual**	**(b) OObjetos**
Líneas totales código	5429	6268	6165
Código personalz. total	-	826	728
Líneas personalz. encapsul.	-	826	572
Líneas personalz. dispersas	-	0	156
% código disperso	-	0	21.4
Nº clases afectadas	-	0	9

Cuadro 7.3: Comparativa en la extensión del código fuente y efecto de enmarañamiento entre las distintas versiones en la componente de personalización.

La tabla 7.4 recoge los resultados obtenidos en relación a las métricas relacionadas con el tamaño del *bytecode*, todos ellos expresados en *Kbytes*. Los valores recopilados son los mismos que en el experimento anterior, y quedan detallados en la tabla siguiendo el mismo orden que en la tabla 7.2.

	Versión original	(a) Aspectual	(b) OObjetos
Tamaño código fuente	144	166,1	163,45
***bytecode* código fuente**	144	277	170
Recursos y librerías	74,4	74,4	74,4
Líbreria AspectJrt	--	204	--
Tamaño total *bytecode*	219	556	245
***bytecode* comprimido**	124	276	138

Cuadro 7.4: Comparativa en el tamaño del bytecode entre las distintas versiones en la componente de personalización.

Tal y como se observa, el tamaño final de la aplicación *aspectual* duplica el de la aplicación *orientada a objetos*. Los motivos son los mismos expuestos para el experimento anterior: la ineficiencia del paso de recomposición y el hecho de tener que incorporar la librería *aspectjrt* para obtener una versión *orientada a aspectos*. A esto se le suma el hecho de que, además, en este caso, la extensión del código *aspectual* también juega en contra. Conviene recordar, no obstante, que la versión *OObjetos* desarrollada es la que resulta más favorable, es decir, la que aplica el patrón *Singleton*.

7.4.4. Discusión acerca de la experimentación llevada a cabo

Los resultados obtenidos para las métricas relacionadas con la extensión del código y el tamaño de la aplicación final no resultan favorables para las versiones *aspectuales*, sobretodo en el segundo experimento realizado. Sin embargo, a pesar de obtener tamaños considerables, las aplicaciones han sido soportadas sin problemas por los dispositivos utilizados en la experimentación. Una vez superada esa preocupación inicial, el centro de atención no está en la extensión del código, sino en el nivel de enmarañamiento e impacto en la aplicación subyacente (reflejados en las líneas cuarta, quinta y sexta de las tablas 7.1 y 7.3), y lo que es aún más subjetivo y complicado de cuantificar: sus consecuentes efectos negativos en la evolución y mantenimiento del código, esto es, en la calidad del software. Cuanto mayor es el impacto en la aplicación mayor es el acoplamiento entre la funcionalidad núcleo y la funcionalidad auxiliar y, en consecuencia, resulta menos factible la reutilización de código. Los criterios de calidad de software resultan difíciles de medir, y tan sólo se ha dispuesto de recursos rudimentarios en la obtención de resultados. No obstante, los valores recogidos han servido para poner de manifiesto que, en efecto, las versiones *aspectuales* están libres de acoplamiento, consiguiendo preservar las propiedades

de transparencia y ortogonalidad, como metas perseguidas mediante la aplicación de la arquitectura software propuesta.

En cuanto a la reutilización del código, ha quedado demostrada también en las versiones *aspectuales*, al poder utilizar la misma aplicación base para construir las distintas versiones utilizadas en los dos experimentos. Sin embargo, en la versión *orientada a objetos*, se ha tenido que volver a recurrir a la versión original para pasar del experimento 1 al experimento 2, puesto que el código ha quedado visiblemente afectado de añadir la componente de regulación del brillo. Eso demuestra la falta de flexibilidad, dificultad de mantenimiento y evolución de las aplicaciones *sensibles al contexto* construidas con una versión estrictamente *orientada a objetos*. Otra observación remarcable es que se incurre en estos problemas a pesar de haber utilizado una versión con patrones (patrón *Observador* y *Singleton*). En efecto, el uso de patrones proporciona modularización, pero requiere la realización de muchos más cambios al código fuente y su estructura general, sobretodo si el comportamiento *sensible al contexto* va siendo añadido incrementalmente, como es el caso de los experimentos llevados a cabo. En definitiva, el impacto sobre el código aumenta.

Por lo que respecta a la reutilización de los componentes *sensibles al contexto* desarrollados con *aspectos*, los que han sido utilizadas en los experimentos son específicas para la aplicación destino ("LectorNoticias"). No obstante, tal y como se presenta en el capítulo siguiente, el *Framework de Plasticidad Implícita* ofrece una versión altamente reutilizable de dichas componentes, con objeto de facilitar su instanciación en distintas aplicaciones.

Cabe destacar que todos los problemas derivados de la falta de modularidad en las versiones *orientadas a objetos* se acentúan conforme más componentes relacionadas con el contexto se van incorporando. En efecto, cada nueva componente enmaraña el código original, el cual debe atender no dos, sino tantas funcionalidades como componentes vayan siendo integradas. Sin embargo, en las versiones *aspectuales* el código base permanece intacto independientemente del número de componentes incorporadas. La facilidad para adaptar la aplicación a distintas versiones *sensibles al contexto* es, por tanto, total, mientras que en las versiones *orientadas a objetos* una variación en las necesidades contextuales puede tener efectos traumáticos.

Por último, en relación a la extensión del código y tamaño de la aplicación final, cuanto mayor es el número de componentes *sensibles al contexto* incorporadas la distancia entre los tamaños resultantes de ambas versiones –*aspectual* y *orientada a objetos*- se va reduciendo. En efecto, una vez incluida la librería *aspectjrt* y generadas las clases extra con capacidades reflexivas, como consecuencia de los problemas derivados de la ineficiencia del *weaver*, la inclusión de nuevas unidades de programa *aspecto* no va a provocar un nuevo incremento significativo en la extensión de la aplicación resultante. Se puede

afirmar que cuantas más unidades *aspecto* son añadidas se obtiene un mayor rendimiento de la inclusión de la librería *aspectjrt*. En consecuencia, la diferencia en tamaño con la correspondiente versión *orientada a objetos* se acorta. De hecho, el tamaño de la aplicación "LectorNoticias" resultante de incorporar todos los componentes presentados (la regulación del brillo de pantalla, juntamente con los componentes de personalización, esto es, el sistema "LectorNoticiasAumentado" descrito en la *sección 7.2.2.*) para ambas versiones así lo demuestra. El tamaño del archivo *jar aspectual* es de 287 Kbytes, frente a los 148,3 Kbytes de la versión *orientada a objetos*, lo que supone un 49 % de aumento frente al 50 % anterior en el segundo experimento.

Bajo la opinión del autor de esta tesis, las ventajas relacionadas con la calidad de software que proporciona la POA para el desarrollo de componentes *sensibles al contexto* resultan más relevantes que los problemas de eficiencia que se derivan de su uso, los cuales, en cualquier caso, pueden ser superados. Cabe esperar que en un futuro próximo, con la aparición de *weavers* y librerías de *AspectJ* específicos para J2ME, todos estos inconvenientes queden resueltos.

Puede consultarse [Vil06] y [SV06] al respecto de los diseños y las pruebas realizadas descritas aquí.

7.5. Resumen y conclusiones del capítulo

A pesar de los inconvenientes y limitaciones derivados del desarrollo de aplicaciones *AspectJ* para dispositivos compactos, las técnicas de POA resultan útiles y satisfactorias para el desarrollo de componentes *sensibles al contexto*, tal y como señalan los trabajos experimentales presentados en este capítulo. En efecto, los dispositivos sobre los que se ha llevado a cabo la experimentación (dispositivos 2G provistos de J2ME de las marcas Nokia, Sony Ericsson y Motorola) han soportado con éxito la ejecución de las aplicaciones resultantes, a pesar de alcanzar un tamaño considerable (principalmente por el hecho de tener que enlazar el código con una librería que no ha sido diseñada específicamente para aplicaciones de este tipo). El no disponer de una versión apropiada de *AspectJ* es un problema cuya solución no está en nuestras manos, aunque todo hace pensar que a corto plazo será ofrecida.

Las conclusiones más relevantes y en las que se desea incidir son las relacionadas con la calidad del software resultante. Como se ha podido comprobar, en las versiones *aspectuales* el nivel de enmarañamiento y, en consecuencia, el impacto sobre la aplicación subyacente es nulo, lo que demuestra que la aplicación base está libre de acoplamiento. La dificultad para variar el código, propia de las versiones *orientadas a objetos*, contrasta con la facilidad

con que en las versiones *aspectuales* puede ser introducido el tratamiento del contexto. La legibilidad de código se mantiene, mientras que en las versiones *orientadas a objetos* ésta se va oscureciendo a medida que se incorporan nuevas componentes. En consecuencia, cualquier variación en las necesidades contextuales puede ser resuelta fácilmente en una versión *aspectual*, mientras que en la versión *orientada a objetos* resulta muy complicado.

Estas observaciones vienen a demostrar que la ortogonalidad y transparencia proporcionada por los *aspectos* va en beneficio de la reutilización de código, premisa que avala y justifica el desarrollo de un framework genérico basado en *aspectos* para el tratamiento del contexto (el *Framework de Plasticidad Implícita*). El disponer de componentes reutilizables e independientes del sistema, sin duda facilita el desarrollo de aplicaciones *sensibles al contexto* construidas bajo las premisas planteadas en la presente tesis (véase *Capítulo 6*; *sección 6.2.1.2.1.*). El proceso es tan simple como escoger qué *aspectos* se desean incluir en la aplicación, con objeto de ser considerados en el paso de *weaving*.

Todas las premisas y propiedades planteadas en el *Capítulo 6* para la arquitectura software han podido ser confirmadas tras su diseño, desarrollo de pruebas y experimentación con dispositivos limitados. Éstas se presentan en detalle en la *sección 7.3.1.1.* de este mismo capítulo. En definitiva, tal y como se preveía desde un principio (véase *Hipótesis 4*; *Capítulo 1 – sección 1.3.*) las técnicas de POA favorecen la integración de los mecanismos de adaptación proactiva en la operativa del sistema bajo unos cánones de calidad.

El estudio de factibilidad realizado en el *Capítulo 6* (véanse *secciones 6.2.2.*, 6,2,3. y 6,2,4.), juntamente con las experimentaciones presentadas en este capítulo constituyen una *Prueba de Concepto*[9] (del inglés *proof of concept*) [Sap04] que ha servido para validar empíricamente la viabilidad de la arquitectura software propuesta en el *Capítulo 6* para el *Motor de Plasticidad Implícita* en el desarrollo de componentes *sensibles al contexto* para aplicaciones móviles. Por otro lado, demuestra la conveniencia de la aplicación de las técnicas de POA para la separación de conceptos, obteniendo con ello los consecuentes beneficios en la calidad de software, entre ellos la versatilidad para dar solución a futuras necesidades contextuales.

Llegado el momento en que se disponga de una versión de *AspectJ* para J2ME (de hecho, ya ha sido anunciada extra-oficialmente), las aplicaciones *sensibles al contexto* no sólo podrán estar notablemente optimizadas, sino que las ventajas derivadas de aplicar POA en este tipo de componentes resultarán plenamente satisfactorias.

[9]Una prueba de concepto es una realización corta y/o sintética (o una sinopsis) de una cierta idea o método, cuyo propósito es el de verificar que un concepto o teoría es factible y puede resultar útil [Sap04]. Las pruebas de concepto son utilizadas en el ámbito de la investigación aplicada, a fin de demostrar la viabilidad de los puntos esenciales de una metodología, métodos o idea, verificando su potencialidad y utilidad, previo a su implementación o implantación. En el ámbito del desarrollo de software, las pruebas de concepto se entienden como un paso previo a la construcción de prototipos.

Capítulo 8

Framework genérico propuesto: el *Framework de Plasticidad Implícita*

"Good programmers know what to write. Great ones know what to use. Exceptional programmers know how to write code that others can use."

E. Raymond

Este capítulo está destinado a describir el *Framework de Plasticidad Implícita* (FPI en lo sucesivo). Como ya se ha mencionado en los capítulos anteriores, se trata de un framework genérico para dotar a las aplicaciones de capacidades de *plasticidad implícita* las cuales, siguiendo el enfoque de la *visión dicotómica*, se integran en el propio sistema, a fin de que éstos se muestren lo más autónomos posible.

En primer lugar, gracias a la experiencia adquirida en el desarrollo de MPIs para los casos de estudio presentados en el capítulo anterior, y a partir de los diseños y esquemas de solución propuestos en sus correspondientes versiones genéricas, descritas en el *Apéndice B*, se presentan las distintas estrategias y consideraciones aplicadas parar incrementar el grado de genericidad. Estas estrategias aportan una mayor reutilización no sólo en distintos sistemas, sino también entre las distintas unidades de programa que conforman los componentes desarrollados. Además, repercuten también en una mayor facilidad a la hora de aplicar el framework en aplicaciones concretas, puesto que se reduce considerablemente la cantidad de código a especializar. A continuación, se presenta en detalle el diseño resultante de todas estas consideraciones, dando lugar a una solución generalizada para las cinco componentes del contexto desarrolladas, las cuales permiten su aplicación en distintos dominios. La explicación va acompañada de los distintos diagramas de clases y *aspectos*, correspondientes a cada una de las capas.

En la última sección se presenta la integración de todos los componentes operativos propuestos o introducidos en la presente tesis: el *Motor de Plasticidad Explícita* (MPE), el *Motor de Plasticidad Implícita* (MPI) y el FPI, a fin de proporcionar una visión integradora de la operativa conjunta bajo el enfoque de la *visión dicotómica de plasticidad*. Se hace especial hincapié en el papel que desempeña el FPI en el proceso llevado a cabo en el *servidor de plasticidad*, así como las etapas en que se divide dicho proceso.

8.1. Estrategias aplicadas para obtener reutilización

Tal y como se ha expuesto a lo largo de este documento, uno de los objetivos propuestos en esta tesis es el de desarrollar un framework genérico formado por un conjunto de componentes semi-completas en las que intervienen clases y *aspectos* convenientemente acoplados entres sí, las cuales incorporan el tratamiento y consecuente adaptación de ciertas *restricciones de tiempo real* de forma genérica. Su misión es la de facilitar la construcción de MPIs para diversos tipos de sistemas y necesidades contextuales, capaces de dotar a las aplicaciones objetivo de *plasticidad implícita*.

Con ese propósito, una vez identificados los esquemas de solución para cada uno de los componentes de adaptación considerados, el siguiente paso para su construcción es el de identificar, reducir y encapsular las dependencias con el sistema recurriendo a la modularización y abstracción de las partes de código que las contienen, lo que significa delegar en subclases o *sub-aspectos* la implementación de esas responsabilidades. El objetivo perseguido es el de ofrecer un framework lo más completo posible generalizando tantas partes de código como sea posible para evitar su recodificación, e identificando las partes que deben ser completadas, particularizadas o especializadas.

Tal y como se analiza en el *Capítulo 7* (véase *Capítulo 7 - sección 7.3.1.1.1.*), las dependencias entre la *capa sensible al contexto* y la *capa lógica* se producen tan sólo puntualmente. No obstante, entre la *capa aspectual* y la *capa lógica* son ineludibles. En esta sección se presentan en detalle las estrategias utilizadas para obtener la genericidad deseada.

8.1.1. Anotaciones de metadatos

Tal y como se introduce en el *Capítulo 6*, la *Programación Orientada a Aspectos* consigue interferir el comportamiento de una aplicación sin causar ningún impacto en el código, respondiendo de ese modo al requisito planteado inicialmente de transparencia. Sin embargo, su aplicación requiere un conocimiento profundo del sistema subyacente, lo que

resulta generalmente en la obtención de diseños específicos del sistema y, en consecuencia, en un fuerte acoplamiento de la *capa aspectual* hacia la *capa lógica*.

Buena parte de esas dependencias se concentran en el establecimiento de los *puntos de unión*, los cuales no escapan de la necesidad de referenciar ciertas piezas de código o elementos del programa base. Esto es así porque la forma más común y simple de establecer *puntos de unión* consiste en recurrir a los propios nombres y signaturas de los elementos de programa de que consta el sistema subyacente (en general, *clases* y *métodos*), lo que se conoce como *signatura basada en método*, incurriendo de ese modo en dependencias insalvables. El diseño aspectual resultante es específico del sistema y altamente acoplado al mismo, lo que supone una fuerte limitación en la genericidad y reutilización, siendo ambas dos metas pertenecientes al planteamiento inicial.

Para evitar esta fuerte dependencia con el sistema y reducir el acoplamiento entre los *aspectos* y las clases de la *capa lógica* se requiere recurrir a otro enfoque para establecer los *puntos de unión*, instaurando, a poder ser, una modalidad genérica. La opción que se propone en esta tesis para alcanzar esa genericidad es utilizar en la definición de los *puntos de unión* una *signatura basada en metadatos*[1], es decir, basada en información adicional que se expresa mediante elementos de programa modificadores llamados *anotaciones*. En efecto, la facilidad de *metadatos* proporciona un mecanismo estándar de adjuntar datos adicionales a los distintos elementos de un programa. Las anotaciones pueden ser añadidas en las declaraciones de paquetes, clases, declaraciones de tipos, constructores, métodos, campos, parámetros y variables. Un ejemplo de anotación que resulta muy familiar es la etiqueta @deprecated. Se pueden marcar distintos métodos con @deprecated para indicar que no deberían ser utilizados. La *anotación* es, por tanto, una etiqueta que se inserta en el código fuente, la cual no altera directamente su semántica, pero afecta la manera en que los programas son tratados por ciertas herramientas y librerías, afectando la ejecución del programa.

En el caso del *Framework de Plasticidad Implícita* se utilizan las *anotaciones de metadatos* para establecer los *puntos de unión*, lo que provoca que sean interceptados aquellos *métodos* o *clases* de la aplicación núcleo que han sido expresamente *'anotadas'* con el *metadato* utilizado en el *punto de unión*. De ese modo se consigue desviar la dependencia de los *aspectos* al nombre de las *anotaciones de metadatos* utilizadas, la cual puede ser conocida de antemano. Para el caso particular tratado aquí (construcción de un framework genérico), esta información debe ser incorporada con el propio framework.

[1]Datos acerca de los datos. En el contexto de un lenguaje de programación, los *metadatos* constituyen información adicional que acompaña ciertos elementos de programa.

En definitiva, las *anotaciones* consiguen eliminar una dependencia directa con el sistema en particular, constituyendo el 'punto de enganche' o anzuelo que permite acoplar (anclar) lo módulos proporcionados por el framework en las aplicaciones. La generalización en el uso de esas mismas *anotaciones* en cualquier sistema que requiera ese mismo tratamiento es la clave para proporcionar reutilización.

Así, al igual que se hace con @deprecated, se introducirían las distintas *anotaciones* a lo largo de las clases y métodos que se desean interceptar. Por lo que respecta a los *aspectos* involucrados, la declaración de *puntos de corte* tomaría este aspecto:

```
pointcut nomPointcut (listaParm): execution(@Brillo * *.*(..));
```

si de los que se trata es de capturar la ejecución de cualquier método que lleve asociada la *anotación Brillo*.

8.1.1.1. Inconvenientes y algunas soluciones

Esta aproximación no evita tener que tener un conocimiento preciso no sólo del código núcleo del sistema -a fin de poder integrar las pertinentes anotaciones estratégicamente en los *métodos* y *clases* que requieren el tratamiento en cuestión-, sino también un conocimiento apropiado de lo que representa cada una de las anotaciones en la operativa del framework. Este último aspecto se aborda mediante una adecuada documentación del framework.

Adicionalmente, la aplicación de este enfoque deja de ser no invasivo y totalmente transparente. La aplicación subyacente sí se ve afectada, pues debe integrar esas anotaciones que, dependiendo del número de *restricciones de tiempo real* a tratar, el volumen de *anotaciones de metadatos* puede llegar a ser considerable y por lo tanto tener un impacto importante. Este fenómeno es conocido con el término inglés *annotation clutter* [Lad03], que podría ser traducido como 'caos de anotaciones'. Obviamente, aunque esta opción resulta interesante en cuanto a la liberación de dependencias del sistema, no responde sin embargo a nuestros intereses relacionados con la preservación de la transparencia.

Existe una solución para evitar este último inconveniente. Consiste en utilizar una clase especial de *aspecto* llamado *aspecto anotador* [Lad03], a introducir específicamente para concentrar y encapsular todas las *anotaciones* de *metadatos* requeridas por el sistema y su correspondiente asociación con los elementos del programa subyacente. Representa un tipo especial de sección declarativa que, sin tener que adjuntar explícitamente las anotaciones a los elementos del programa 'in situ', logra establecer igualmente su asociación con los

mismos. En definitiva, esta componente permite liberar completamente al código base de tener que embeber ese tipo de información adicional. Además, incluso, puede ser integrado en la propia *capa aspectual*, con lo que el sistema base permanece ajeno al hecho de que está siendo aumentado.

Un ejemplo de aspecto anotador sería el siguiente:

```
public aspect Anotador {
   declare @method:
     public void nomClase.nomMétodo(listaParm):@Brillo();
}
```

Este *aspecto anotador* designa una *anotación* denominada *Brillo* en un método de nombre *nomMétodo* perteneciente a una clase de nombre *nomClase*. Si se desea suministrar esa *anotación* a un método constructor la declaración es la siguiente:

```
declare @constructor: nomClase.new(listaParm): @Brillo();
```

asignándose esa misma *anotación* al método contructor de la clase *nomClase*. También existen las declaraciones *declare @field*, *declare @type* y *declare @package*, de acuerdo al destino de la *anotación*.

8.1.1.2. Generalización en su aplicación práctica: definición de tipos para las anotaciones

A pesar de la potencialidad ofrecida por la utilización de *anotaciones de metadatos* en la definición de los *puntos de corte*, aún se requiere aplicar un paso más para que esta técnica pueda ser aplicada de forma generalizada en la construcción del *Framework de Plasticidad Implícita*.

Como se ha podido observar de las soluciones aportadas para los casos de estudio, y fijándonos en una componente concreta, cada uno de los *aspectos* principales de la *capa aspectual* incorpora más de un *punto de corte*, a efectos de repartir apropiadamente toda la funcionalidad a inyectar en distintos puntos de la ejecución del sistema. Si nos limitamos a marcar todos los métodos a interceptar por todos los *puntos de corte* correspondientes a una componente con una misma etiqueta, la interpretación para los mismos será idéntica, no obteniendo por tanto el efecto esperado -en lugar de repartir la funcionalidad ésta se ejecutaría por completo en un solo punto. Por otro lado, la alternativa consistente en definir tantas anotaciones como *puntos de corte* sean necesarios en cada componente introduciría

confusión y complejidad en su aplicación práctica. Así, por ejemplo, si de lo que se trata es de incorporar una componente de apoyo al trabajo en grupo, lo adecuado es que se utilice una única anotación *(SensibilidadGrupo)* y no tantas como *puntos de corte*. En particular, en esta componente se tendría que recurrir a un total de cinco *anotaciones* distintas.

Para solucionar esta problemática J2SE 5.0 permite personalizar las *anotaciones* definiendo tipos *anotación*, pudiendo declarar sus propios miembros. De ese modo la especificación de una cierta *anotación* puede ir acompañada de distintos pares miembro-valor -tanto en la aplicación base como en las unidades de programa *aspecto*-, a fin de distinguir distintas connotaciones en el tratamiento llevado a cabo en una misma componente y, en definitiva, poder repartir el código de los *consejos* a lo largo de toda la aplicación.

En el FPI se ha definido un solo miembro de tipo *String* para todas las *anotaciones*, a efectos de que el correspondiente valor cadena sea lo más significativo posible para identificar qué punto de la ejecución se desea interceptar en cada caso.

Para declarar un tipo para la anotación Brillo se declararía la siguiente componente:

```
@Retention(value=RetentionPolicy.RUNTIME)
 public @interface Brillo {
    String valor();
 }
```

donde en la primera línea se está indicando que la *anotación Brillo* debe ser retenida en tiempo de ejecución para poder ser procesada por la JVM.

La designación de *anotaciones* provistas de atributos a través del *aspecto anotador* tomaría esta forma:

```
declare @method:
      public void *.nomMétodo(lParm): @Brillo(valor="Intenso");
```

Mediante esta línea de código se está asociando la *anotación Brillo* con el valor "Intenso" para el atributo 'valor' al método *nomMétodo*. Esta asociación se introduce a través del *aspecto anotador*.

La definición de un *punto de corte* que intercepta todos aquellos métodos provistos de la *anotación Brillo* con valor "Intenso" en su atributo 'valor' toma este aspecto:

```
pointcut nomPointcut(Brillo b): execution(* *.*(..)) &&
             @annotation(b) && if (b.valor=="Intenso");
```

El designador *if* filtra los métodos con valor "Intenso" asociado.

8.1.1.3. Tecnología de soporte

Esta combinación de técnicas de programación está disponible en el mudo Java, aunque cabe remarcar que el uso de *anotaciones* tan sólo se incluye en la modalidad J2SE de java. La versión J2ME para dispositivos compactos no las soporta. Por lo tanto, es necesario esperar a una nueva versión de J2ME para poder obtener los mismos beneficios que en J2SE.

En particular, esta limitación ha repercutido en la elaboración de dos diseños para cada componente: una para aplicaciones de sobremesa tradicionales –a las que se les hace referencia como 'aplicaciones fijas'- en J2SE, y otra para aplicaciones móviles en J2ME, en el que, por supuesto, el grado de reutilización se ve considerablemente reducido al no poder recurrir a esta estrategia.

Por su parte, y como ya se introduce en el *Capitulo 6*, el lenguaje de POA adoptado es el lenguaje *AspectJ* [Lad05] (véase *Capítulo 6*; *sección 6.2.5.*) que, además de ser una extensión del lenguaje Java relativamente fácil de aprender, entre sus muchas ventajas soporta también la definición de *puntos de corte* basados en *metadatos*. En particular, no ha sido hasta la versión 5.0 de Java que las *anotaciones de metadatos* han sido incorporadas en el lenguaje núcleo. Con respecto al lenguaje *AspectJ* éstas se han incorporado en la versión 5, la cual permite interpretar correctamente las nuevas características de Java 5.0.

Se puede consultar una introducción más detallada del lenguaje en el *Apéndice A*.

8.1.1.4. Conclusiones respecto al uso de la técnica descrita

Indudablemente, la combinación de las *anotaciones de metadatos* con los *aspectos*, juntamente con el uso de un *aspecto anotador*, reporta una ventaja clara: los *aspectos* que incorporan la nueva funcionalidad al sistema base –a los que referir como *aspectos* principales- son abstraídos del conocimiento necesario acerca del sistema, puesto que los *puntos de unión* correspondientes son expresados en términos de *anotaciones*, en lugar de hacerlo en términos de métodos y clases. Esta particularidad es especialmente interesante para la reutilización. En efecto, para que un mismo *aspecto* pueda ser operativo en otro sistema distinto tan sólo se requiere asociar esas mismas *anotaciones* con el nuevo sistema recurriendo a un *aspecto anotador*, evitando con ello alterar el propio código *aspectual*[2], además de preservar la propiedad de transparencia en el sistema base. En efecto, buena parte de la adaptación de una misma componente del contexto a un sistema determinado

[2]el código de los *aspectos* principales está expresado en base a unas mismas *anotaciones*, independientemente del sistema con el que están asociadas.

consiste en personalizar el *aspecto anotador* al nuevo sistema, constituyendo el anclaje para 'enganchar' el tratamiento de las necesidades contextuales con el sistema subyacente.

Aunque la asignación de *anotaciones de metadatos* al sistema subyacente no constituye una tarea trivial, permite sin embargo aislar las dependencias del sistema de la propia definición de los *aspectos*. Esto, a su vez, permite adaptar fácilmente un mismo sistema a distintas necesidades contextuales o mecanismos de adaptación, puesto que las dependencias de los *aspectos* con el sistema base se concentran en un *aspecto* separado que hace la función de 'suministrador de anotaciones'.

Por último, la posibilidad de generalización que aporta la definición de tipos para las *anotaciones* aporta un uso elegante de las *anotaciones de metadatos* en el framework. En este sentido, el framework ofrece un auténtico repositorio de anotaciones personalizadas, por lo que resulta indispensable proporcionar documentación al respecto, a fin de dar a conocer el propósito de cada *anotación*.

8.1.2. Herencia y patrones específicos de POA

La aplicación de la relaciones de generalización entre *aspectos*, así como de patrones específicos de POA, sin duda, facilitan el tratamiento y aislamiento de las dependencias con el sistema subyacente, con objeto de alcanzar el nivel de genericidad y reutilización perseguido. Básicamente, la idea consiste en concentrar y encapsular la parte de código específica del sistema en construcciones de programa tales como métodos y *puntos de corte* declarados como abstractos en un *aspecto* abstracto, con el fin de delegar en los *sub-aspectos* la concreción de esas particularidades.

En general, existen tres puntos en la definición de una unidad *aspecto* propensos a contener dependencias. Son los siguientes: definición de los *puntos de unión*, definición de los *puntos de corte* y definición de los *consejos*.

8.1.2.1. Definición de los puntos de unión

Conceptualmente, los *puntos de unión*, al ser la parte de código que establece los puntos concretos en el flujo de ejecución de un programa a ser interferidos por los *aspectos*, están destinados irremediablemente a contener referencias concretas del sistema base, y por lo tanto, constituyen un foco de dependencias. El uso de los comodines (*wildcards*) para identificar patrones en la signatura de los métodos no siempre resulta factible.

Esta problemática es debida a que la definición de los *puntos de unión* suele estar basada en la signatura de los métodos, lo que establece un acoplamiento ineludible con el

código subyacente. Precisamente la técnica propuesta para superar esta limitación consiste en sustituir la signatura habitual por una signatura basada en *anotaciones de metadatos*, siguiendo el enfoque que se presenta en la sección anterior. Las *anotaciones de metadatos* ofrecen una forma genérica de referenciar piezas de código del sistema base, aplicable a cualquier sistema que necesite incorporar una determinada funcionalidad a su operativa básica.

8.1.2.2. Definición de los puntos de corte

En la definición de los *puntos de corte* se pueden encontrar tres tipos de dependencias o particularidades, las cuales responden a tres necesidades de especialización distintas. Son las siguientes:

8.1.2.2.1. Especificación del designador del punto de corte

El designador o tipo del *punto de corte* es la palabra clave propia del lenguaje que establece qué tipo de operación del sistema base se desea interferir (e. g. *call*, *execution*, *set*, *get*, *within*, *withincode*, etc.), tal y como puede ser consultado en el *Apéndice A*.

En ocasiones, el designador no puede ser establecido de antemano, sino que, dependiendo de la aplicación, la matización entre por ejemplo usar un *call* o un *execution* es clave, y por tanto constituye también una particularidad propia del sistema.

Siempre y cuando el *consejo* asociado a este *punto de corte* esté libre de dependencias, la opción más acertada en estos casos consiste en aplicar un *idiom*[3] de *AspectJ* conocido como *abstract pointcut* [HUS03].

Abstract pointcut idiom

Aplicable cuando el comportamiento del *aspecto* puede ser completamente especificado, esto es, el *consejo* está libre de dependencias, pero se desconoce cuándo el comportamiento extra debe ser puesto en funcionamiento. Este patrón también es válido cuando el *punto de unión* es dependiente del sistema, si lo que se pretende es obtener código reutilizable, como en este caso. En términos generales, esta opción es válida siempre que, a pesar de que no se desea o no se puede particularizar la definición de los *puntos de corte*, el código asociado al *consejo* está libre de esa restricción. En efecto, a pesar de definir el *punto de corte* como abstracto, el *consejo* asociado puede estar implementado en el propio *aspecto* base. La figura 8.1 representa este patrón.

[3]patrón específico de un lenguaje de programación.

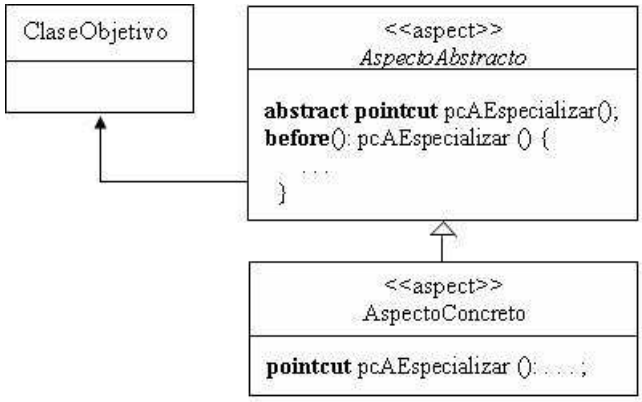


Figura 8.1: *Abstract pointcut* idiom.

Tal y como se observa, la aplicación de este patrón obliga a recurrir a la definición de un *sub-aspecto*, a pesar de que sólo sea necesario definir un *punto de corte*. Esto reduce en cierto modo el grado de reutilización del framework. Por otro lado, el establecimiento de un *punto de corte* requiere cierto conocimiento del lenguaje de POA, en este caso de *AspectJ*.

La aplicación de una signatura basada en *anotaciones de metadatos* evita la necesidad de recurrir a la definición de un *punto de corte* abstracto y, por lo tanto, a la especialización. Si esta estrategia es extensible a todos los *puntos de corte* de un cierto *aspecto* el resultado es el de un *aspecto* totalmente reutilizable. En consecuencia, otro de los beneficios del uso de la estrategia basada en *anotaciones* es, por tanto, el grado de reutilización obtenido, así como cierta liberación en el nivel de conocimiento de las técnicas de POA. Por lo que respecta al nivel de conocimiento requerido acerca del código del sistema subyacente, éste es el mismo en ambos casos, puesto que igualmente debe identificarse qué pieza de código (método, clase, paquete o porción de código) particular debe ser interceptada. No obstante, la especificación del *punto de unión* varía significativamente. En el caso de aplicar el enfoque basado en *metadatos* se incorpora una *anotación* en el punto oportuno (a encapsular en el *aspecto anotador*). Si se aplica el *abstract pointcut* idiom, el *punto de unión* se concreta en el *sub-aspecto*.

8.1.2.2.2. Argumentos del punto de corte

En muchas ocasiones se requiere capturar el contexto de la ejecución del sistema base (uso de los designadores *this*, *target* o *args*; véase *Apéndice A*). Lo habitual es que los objetos capturados del contexto sean dependientes del sistema, puesto que hacen referencia a objetos de las clases específicas de la aplicación subyacente.

Este tipo de dependencias afectan no sólo en la definición del *punto de corte*, sino también a la definición del *consejo* asociado. Eliminar esta dependencia del *aspecto* base implica redefinir ambas partes (*punto de corte* y *consejo* asociado) en los *sub-aspectos*. La inclusión de un *punto de corte* abstracto en el *aspecto* base resulta útil para dar a conocer que esa parte queda pendiente de definir, dando a conocer la existencia de una parte dependiente del sistema.

El recurrir a la definición de *puntos de corte* abstractos permite identificar y encapsular las dependencias. No obstante, esta opción limita claramente la reutilización a la hora de especializar el *sub-aspecto*. Una posible medida intermedia entre el grado de reutilización y el grado de genericidad es la que se describe a continuación.

Conversión de una dependencia del sistema a una dependencia de la plataforma

En el caso en que el objeto a ser capturado del contexto se trata de una clase que proviene de la jerarquía de clases proporcionada por una API estándar –como por ejemplo la MIDP 2.0 de J2ME-, se puede llegar a una solución intermedia que preservaría la reutilización a cambio de introducir una dependencia de la plataforma que, al fin y al cabo, resulta menos restrictiva que una dependencia del propio sistema.

En definitiva, en lugar de exponer el nombre de la clase específica del sistema en la definición del *punto de corte*, se trata de utilizar el nombre de la super-clase de la que deriva, perteneciente a la API utilizada. En consecuencia, el acoplamiento introducido es con respecto a la API.

Ejemplos son, en la definición de aplicaciones J2ME, referirse a la clase *MIDlet* (paquete *javax.microedition*, propia del perfil MIDP) en lugar de a la clase principal de la aplicación móvil (*ControlObrasMIDlet* o *LectorNoticiasMIDlet* en los casos de estudio –subclases de MIDlet-). Otro ejemplo es el de utilizar las clases Form, TexBox, List, etc., propias de la interfaz (paquete *javax.microedition.lcdui*), en lugar de las pantallas propias del sistema, las cuales derivan de éstas.

De cara a ofrecer un framework independiente de plataforma, sería interesante ofrecer una jerarquía de *aspectos* para cada plataforma. Con ello se cubriría la reutilización no sólo para la plataforma MIDP, sino para cualquiera que haya sido prevista en el framework.

La combinación de esta estrategia junto con la designación de *puntos de unión* basados en *metadatos* proporciona muchas posibilidades para la reutilización de *aspectos*. La opción adoptada en el framework se presenta en la *sección 8.2.4*.

8.1.2.2.3. Definición de puntos de corte condicionales

Otra de las necesidades habituales en la intercepción de código base es el cumplimiento de ciertas condiciones. En general, si lo que se busca es interceptar ciertas circunstancias, lo más factible es construir un *punto de corte* condicional (uso del designador *if*; véase *Apéndice A*).

No obstante, si esas condiciones implican objetos del sistema base, la inclusión de referencias a dichos objetos específicos será inevitable.

Para abstraer las dependencias en las condiciones lo más adecuado es aplicar otro idiom de *AspectJ* denominado *pointcut method* [HK02]. Este idiom lo que hace es encapsular las expresiones condicionales en métodos definidos como abstractos, los cuales retornan el valor lógico correspondiente a la evaluación de la condición encapsulada. En los *sub-aspectos* tan sólo se requiere definir el método en cuestión. El *punto de corte* queda definido como parte del *aspecto* base, evitando tener que ser declarado en el *sub-aspecto* [SV06].

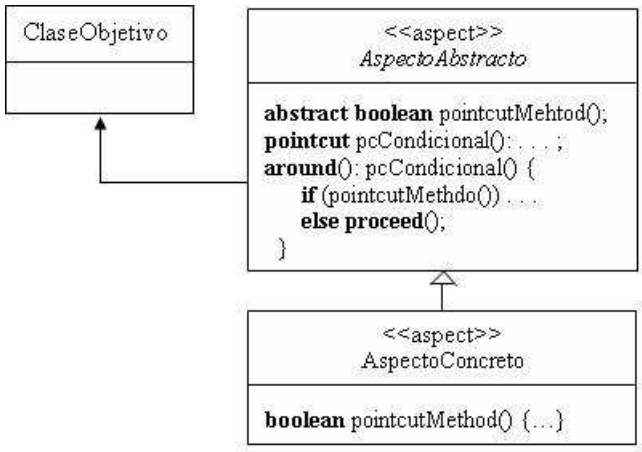


Figura 8.2: *Pointcut method* idiom.

8.1.2.3. Definición de los consejos

Si la parte del *aspecto* que contiene dependencias es el propio *consejo*, es necesario recurrir a distintos pasos de refactorización, así como al uso de otros patrones e *idioms* específicos de POA.

En particular, cuando el *consejo* se compone de una parte fija –esto es, independiente del sistema- y una parte variable –dependiente del sistema-, el *idiom* adecuado es el denominado *template advice* [HK02], el cual se equipara con el conocido patrón GoF *template*

method para POO [GHJV95]. La diferencia entre ambos es que la plantilla es especificada sobre el código de un *aspecto*, en lugar de sobre el código de un método.

Template advice idiom

Patrón que se utiliza cuando el comportamiento *aspectual* –el del *consejo* asociado- presenta ligeras variaciones que se desea encapsular. Por lo tanto, el propósito de este patrón es reutilizar las partes esenciales del código de un *consejo*.

Se utiliza la parte fija de la pieza de código como una plantilla, y la parte variable (operación primitiva) se encapsula en un método abstracto, combinando ambas partes en un *aspecto* abstracto. Dichas operaciones primitivas se definen en *aspectos* concretos, con objeto de especializar cada necesidad o característica específica del sistema, quedando de ese modo encapsuladas las dependencias del sistema [HUS03].

Se trata de una opción muy utilizada y elegante para el diseño de frameworks, puesto que fomenta la reutilización de las parte de código independientes del sistema. Además, la parte que queda pendiente de definir no conlleva codificar ninguna construcción específica de POA, pues la definición de métodos concretos es meramente *Programación Orientada a Objetos* [HUS03]. La figura 8.3 muestra este patrón.

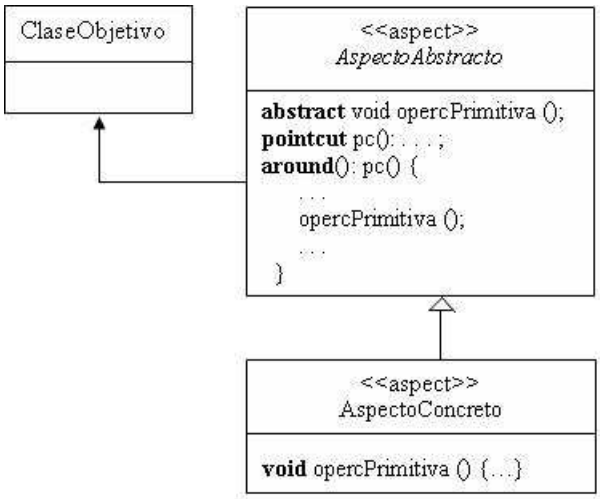


Figura 8.3: *Template advice* idiom.

8.1.3. Definición de métodos constructores para los aspectos

Como se ha podido observar de los casos de estudio, en muchas ocasiones una de las responsabilidades de un *aspecto* de la *capa aspectual* es la de llevar a cabo una operación

de inicialización, ya sea la de crear o inicializar alguna estructura de datos de la *capa sensible al contexto* (por ejemplo, en los objetivos de personalización de una pantalla de ambos casos de estudio se instancía la tabla de los usos en el código de los *aspectos*), o en otros casos la de lanzar a ejecución un *thread* (caso de estudio *'Lector de Noticias'*; componente de adaptación al entorno). En estos casos, en lugar de interceptar algún punto correspondiente al inicio de la ejecución del programa –tal y como se propone tanto en las soluciones aportadas en el *Capítulo 7* como en la versión genérica presentada en el *Apéndice B*-, una opción más reutilizable es evitar esa nueva dependencia llevando a cabo la operación de inicialización en el propio método constructor del *aspecto*. En efecto, esta estrategia evita introducir un *punto de corte* específicamente para ello, y por lo tanto una nueva dependencia.

8.1.4. Declaraciones inter-tipo

En buena parte de las soluciones aportadas en los casos de estudio se ha recurrido al uso de declaraciones inter-tipo para ampliar los métodos ofrecidos por las clases del sistema base involucradas en el proceso de adaptación –las clases que representan las *entidades objetivo* propias de cada sistema-, a fin de poder contar con esa funcionalidad en el código de los *aspectos*. En estos casos se ha recurrido al uso de un *aspecto* específico, al que se le ha denominado *IncrementadordeOperaciones*, encargado justamente de introducir esa nueva operativa en las clases del sistema base. Como este tipo de declaraciones son estrictamente dependientes del sistema, de este modo se consigue su aislamiento respecto al resto del código *aspectual*.

En la construcción del framework se han explotado aún más las posibilidades ofrecidas por las declaraciones inter-tipo, con el propósito de sustituir las referencias a las clases concretas del sistema base a lo largo del código del framework por referencias a clases genéricas –esto es, un mismo nombre de clase, independientemente del dominio de la aplicación. La relación de dichas clases genéricas con las correspondientes clases efectivas, propias de la aplicación, se introduce mediante el uso de declaraciones inter-tipo (en concreto *declaraciones de parentesco*). Esta opción implica introducir cambios más profundos en la estructura estática del sistema base, y en particular de su estructura de clases. Se trata de incorporar un nivel superior en la jerarquía de clases del sistema base involucradas. Se tratan por tanto de una relación de generalización.

Se introduce, por tanto, una super-clase –concretamente se trata de las que aparecen con el nombre de *EntidObjetBase* en el framework; véase *sección 8.3.*- para cada una de las clases específicas del sistema –*entidades objetivo*- involucradas en el tratamiento de la nueva funcionalidad. La sustitución de las referencias a las clases específicas de la

aplicación por referencias a las clases genéricas expresamente introducidas en el código de los componentes de adaptación proporciona la independencia de clases perseguida, incrementando en gran medida el grado de reutilización.

Estas referencias pueden aparecer no sólo en (1) el código de los *consejos* y en (2) los parámetros de los métodos, sino también en (3) los argumentos de los *puntos de corte* (aspecto tratado en la *sección 8.2.2.2.2.*), constituyendo en este último caso, una alternativa a la de declarar los *puntos de corte* como abstractos. Además, la otra alternativa consistente en convertir una dependencia del sistema en una dependencia de la plataforma, presentada también anteriormente, tan sólo resulta válida para aquellas clases que derivan de las clases proporcionadas en una API, no resolviendo las dependencias con las clases propias del dominio del sistema. En realidad, la opción propuesta puede considerarse como una generalización de ésta donde, en lugar de utilizar una jerarquía de clases proporcionada por una cierta API, se aplica una jerarquía particular, 'fabricada' por los propios *aspectos* del framework. Bajo este punto de vista, el framework se concibe como una API.

Las ventajas que aporta esta estrategia son las siguientes:

1. Al estar las clases afectadas del sistema base –en lo sucesivo clases efectivas- en la línea de herencia de la super-clase incorporada –la denominada *EntidObjetBase*-, toda referencia a la clase efectiva puede ser sustituida por una referencia a la correspondiente clase genérica. Es en tiempo de ejecución que ésta es instanciada por la clase efectiva del sistema base, pasando a ser la clase operativa. Con ello se consigue que los *aspectos* principales, encargados de llevar a cabo el tratamiento de las *restricciones de tiempo real* en cada uno de los componentes del framework, puedan liberarse de ese tipo de dependencias.

2. La declaración de una clase genérica permite, adicionalmente, incorporar nuevos métodos a ser heredados por las sub-clases –las entidades efectivas-; métodos que se conoce de antemano que van a ser requeridos en el código de la componente.

El salto hacia un mayor grado de reutilización que proporciona esta estrategia es decisivo de cara a compensar el esfuerzo de aplicar el framework. De hecho, hasta ahora las dependencias introducidas en la definición de *puntos de corte* con argumentos a las clases específicas del sistema no habían sido resueltas.

Todas las declaraciones inter-tipo, tanto las encargadas de incorporar nuevos métodos en las clases efectivas, como las que establecen relaciones de generalización con las nuevas clases genéricas, definidas en la *capa sensible al contexto*, se agrupan en un solo *aspecto* específico para este fin. El nombre utilizado para el mismo es *IncrementadorEstructura*,

que viene a sustituir al que se había denominado en los diseños presentados en el *Capítulo 7* denominado *IncrementadordeOperaciones*.

Adicionalmente, en el caso de utilizar *anotaciones de metadatos* –componentes destinadas a aplicaciones fijas basadas en J2SE-, se incluyen también en un mismo *aspecto* las asociaciones de *anotaciones de metadatos* con el código base, a través del uso de un *aspecto anotador* (véase *sección 8.2.1.1.*). El nombre utilizado en este caso para este *aspecto* en el framework es *AnotadorIncrementador*.

8.2. Descripción del *Framework de Plasticidad Implícita*

Una vez presentadas las versiones genéricas para los componentes trabajados en los casos de estudio del *Capítulo 7* y tras mostrar las técnicas a las que se ha recurrido para relegar las dependencias con el sistema en un segundo plano, a continuación se expone cómo han sido aplicadas cada una de ellas en cada componente con objeto de incrementar el nivel de genericidad, y por lo tanto avanzar un paso en la reutilización del código. Los diseños descritos a continuación constituyen la versión actual del FPI.

8.2.1. Generalización de los componentes de personalización de una pantalla

El framework proporciona dos versiones para esta componente: la versión para aplicaciones móviles en J2ME y la versión basada en J2SE, donde la principal diferencia es la utilización o no de *anotaciones de metadatos*.

8.2.1.1. Capa sensible al contexto

En esta capa se requiere mantener tanto en la memoria volátil como en la memoria persistente el registro del número de ocasiones en que el usuario ha optado por cada uno de los patrones de actuación observados. Dependiendo del objetivo de personalización a tratar y del tipo de pantalla, el patrón de actuación a ser observado implica acciones distintas. Así, cuando se trata de personalizar los valores por defecto de un formulario se recoge la combinación de parámetros introducidos por el usuario. Cuando de lo que se trata es de personalizar una lista se recogen las distintas opciones seleccionadas por el usuario.

En definitiva, tal y como se expone en el *Apéndice B*, para soportar esta información en memoria se ha optado por recurrir a dos clases: la *UsoEntdObj*, y la *TablaUsosEntdObj*.

Por lo que respecta a esta capa, el FPI presenta dos variaciones principales respecto a la versión genérica presentada en el *Apéndice B*. Son las que se presentan a continuación.

8.2.1.1.1. Introducción de clases genéricas

La definición e introducción de clases genéricas, las cuales son utilizadas para referenciar a las *entidades objetivo* manejadas en el sistema base -las *clases efectivas*-, independientemente de cuáles sean éstas. Para ello se recurre al uso de declaraciones inter-tipo, tal y como se propone en la *sección 8.2.4*.

Esta estrategia es aplicada también para referenciar de forma genérica a las pantallas involucradas. Los nombres utilizados son *FormObjet* y *ListObjet* para formularios y listas respectivamente.

Las clases genéricas introducidas son *EntdObjetBase* y *FormObjetBase* como super-clases de *EntidObjet* y *FormObjet* respectivamente en el objetivo de personalización 'valores por defecto en un formulario', y adicionalmente la clase *ListObjetBase* como super-clase de *ListObjet* en el objetivo de personalización 'ordenación de una lista'.

8.2.1.1.2. Construcción de jerarquías reutilizables

Se ha podido observar de los casos de estudio que la necesidad de mantener y gestionar las ocurrencias de las distintas *entidades objetivo* asociadas a los patrones de actuación del usuario es común en los dos objetivos de personalización de pantalla trabajados. Además, esta parte de funcionalidad es extensible a otros casos en los que igualmente se esté interesado en personalizar una pantalla conforme al patrón de actuación del usuario.

Ese es el motivo por el que se ha establecido una jerarquía tanto para las clases de los usos como para la clase de las tablas, con objeto de desglosar e identificar las partes de la funcionalidad que son comunes para cualquier objetivo de personalización de una pantalla con este tipo de necesidades, de las partes específicas para cada objetivo y heurística concreta. Ambas jerarquías se detallan a continuación.

Jerarquía para las tablas

Por lo que respecta a la jerarquía de clases para las tablas, la parte común a diversas heurísticas y objetivos de personalización de pantalla de estas mismas características se reúne en la super-clase de la jerarquía de las tablas, denominada *TablaUsosEntidObjtBase*. Las partes específicas, dependientes de la heurística seleccionada y del objetivo de personalización en particular, quedan definidas en las correspondientes sub-clases. En particular, para los dos objetivos de personalización trabajados con sus respectivas heurísticas, se han definido las clases *TablaUsosEntdObjBaseconModa* y *TablaUsosEntdObjBaseComparable*, tal y como se observa en la figura 8.4 (2 págs. adelante).

Es importante remarcar que la implementación de los métodos se realiza en todo momento utilizando la clase genérica específicamente introducida para hacer referencia a la *entidad objetivo* –la denominada *EntdObjetBase*-, a fin de independizar el código de las particularidades del sistema base. En consecuencia, ambas clases *TablaUsosEntdObjBaseconModa* y *TablaUsosEntdObjBaseComparable* son reutilizables a distintos sistemas en los que se desee aplicar el mismo objetivo de personalización y la misma heurística.

Se requiere, por tanto, un tercer nivel en la jerarquía de las tablas en el que, para un objetivo de personalización y una heurística dados, se determine cuál es la *entidad objetivo* propia de cada sistema a gestionar. Se trata de las clases *TablaUsosEntdObjconModa* y *TablaUsosEntdObjComparable* para los dos objetivos de personalización trabajados, de las que tan sólo se requiere codificar la constructora de la clase. En efecto, el código heredado de las clases padre no requiere ningún tipo de especialización. Es a través del método constructor que se realiza la instanciación de la tabla en la *entidad objetivo* propia de cada sistema, haciendo de ese modo operativas todas las referencias utilizadas en las clases superiores. La figura 8.4 (pág. siguiente) muestra esta jerarquía.

Jerarquía para los usos

Por otro lado, se define también una jerarquía para los usos, suficientemente genérica como para poder aplicar diversas heurísticas.

Al igual que se he realizado para las tablas, se ha establecido una jerarquía con el mismo propósito anterior. La parte común se reúne en la super-clase denominada *UsoEntidObjBase*.

En particular, para el objetivo de personalización 'ordenación de una lista', en la versión destinada a aplicaciones móviles, se requiere implementar una interfaz que deben cumplir las clases a ser ordenadas: la interfaz *Comparable*. Como este caso es particular a ciertos objetivos de personalización y plataformas, se ha optado por incorporar este requisito extra –cumplimiento de la interfaz *Comparable*- definiendo una sub-clase de *UsoEntidObjBase*: la denominada *UsoEntdObjBaseComparable*.

Esta jerarquía se establece también tomando siempre como referencia la clase genérica específicamente introducida para la *entidad objetivo* –la denominada *EntdObjetBase*, consiguiendo que las clases *UsoEntidObjBase* y *UsoEntdObjBaseComparable* sean reutilizables a distintos sistemas.

Un último nivel en la jerarquía de los usos determina cuál es la *entidad objetivo* propia del sistema a gestionar. Se trata de la clase *UsoEntdObj*[4]. En este nivel de la jerarquía se

[4]obviamente en el diagrama de clases de la figura 8.5 aparecen dos nombres distintos para estas clases.

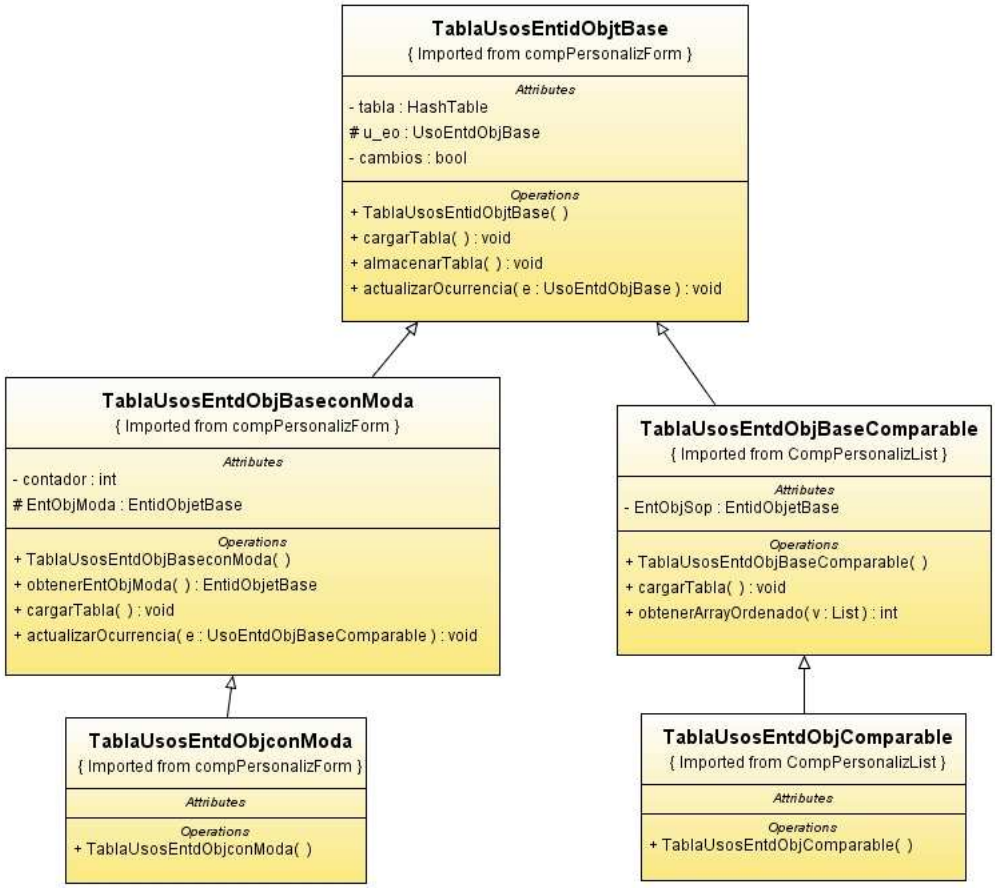


Figura 8.4: Jerarquía de las *tablas de usos* para los componentes de personalización de pantallas en el FPI.

requiere codificar tanto la constructora de la clase como el método *clone*[5] –interfaz *Cloneable*. El código heredado de las clases padre, basado en referencias a la clase genérica, no requiere ningún tipo de especialización. En el método constructor se realiza la instanciación de la clase de los usos en la *entidad objetivo* propia de cada sistema.

La figura 8.5 (pág. siguiente) muestra esta jerarquía.

En el *Apéndice C* se muestra el diagrama de clases en el que se reúnen ambas jerarquías (*tablas de usos* y *usos*), a fin de plasmar las interrelaciones entre las mismas para la construcción de componentes de personalización de pantallas teniendo en cuenta diversas heurísticas y los dos objetivos de personalización propuestos.

[5]El método *clone* crea y retorna copias de los objetos propietarios, evitando tener que conocer en tiempo de compilación la clase concreta de la jerarquía a la que pertenece el objeto. Este aspecto aporta genericidad también en lo que respecta a la creación de objetos.

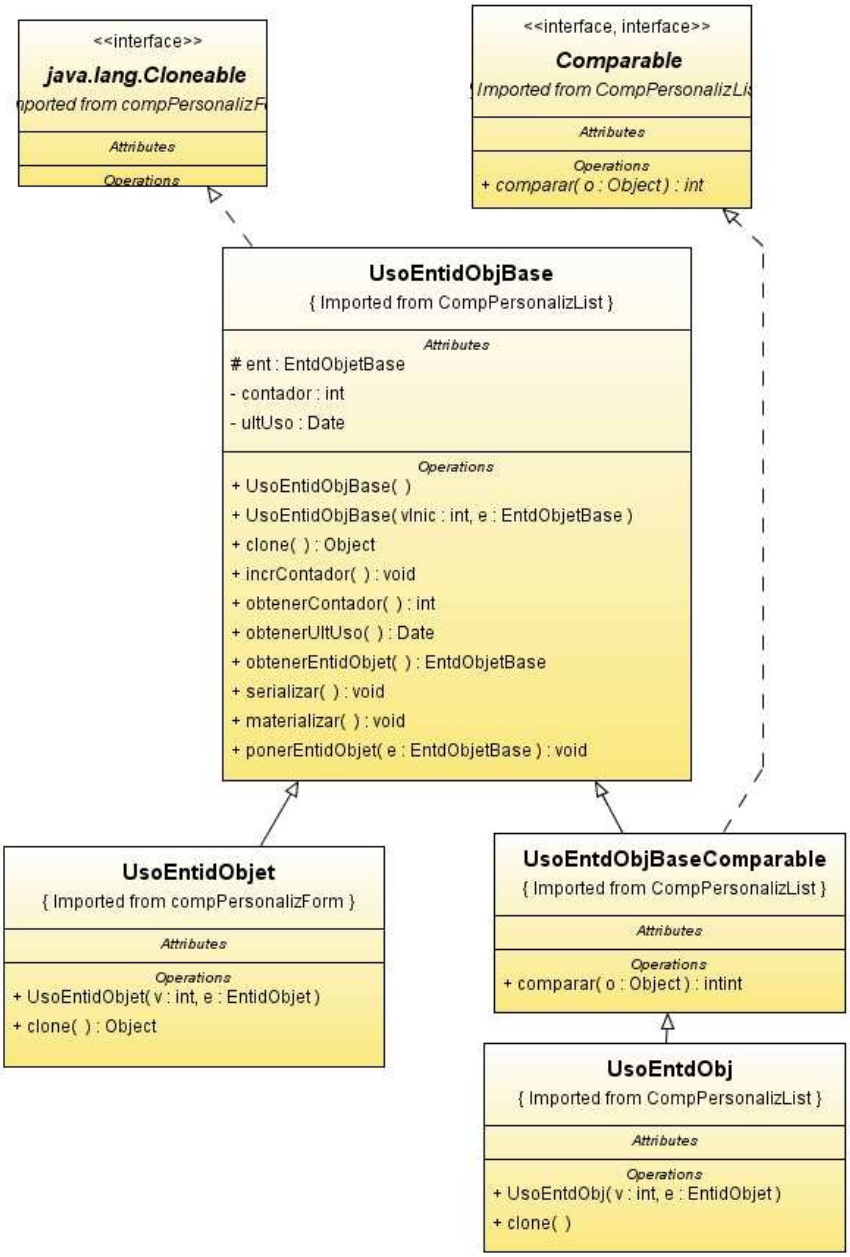


Figura 8.5: Jerarquía de los *usos* para los componentes de personalización de pantallas en el FPI.

8.2.1.1.3. Diagrama de clases de la capa sensible al contexto

A continuación se presentan los diagramas de clases que representan la *capa sensible al contexto* para los dos objetivos de personalización tratados, donde se han introducido tanto las clases genéricas como las jerarquías reutilizables expuestas anteriormente. En particular, la figura 8.6 es la que representa el objetivo de personalización '*valores por defecto en un formulario'*.

En el objetivo de personalización 'ordenación de una lista' se presentan dos diseños que distinguen entre la utilización de un dispositivo móvil o una plataforma fija, cuya diferencia está en la incorporación de las clases que implementan el algoritmo de ordenación en la versión para J2ME. Son los que se representan en las figuras 8.7 y 8.8. En ellos se integran todos los detalles proporcionados en la *sección 8.1.1*.

Objetivo de personalización 'valores por defecto en un formulario'

Véase figura 8.6 (pág. siguiente).

Las clases *EntidObjetBase* y *FormObjetBase* son las clases genéricas expresamente introducidas para aportar genericidad en las referencias a la *entidad objetivo* y el formulario implicado.

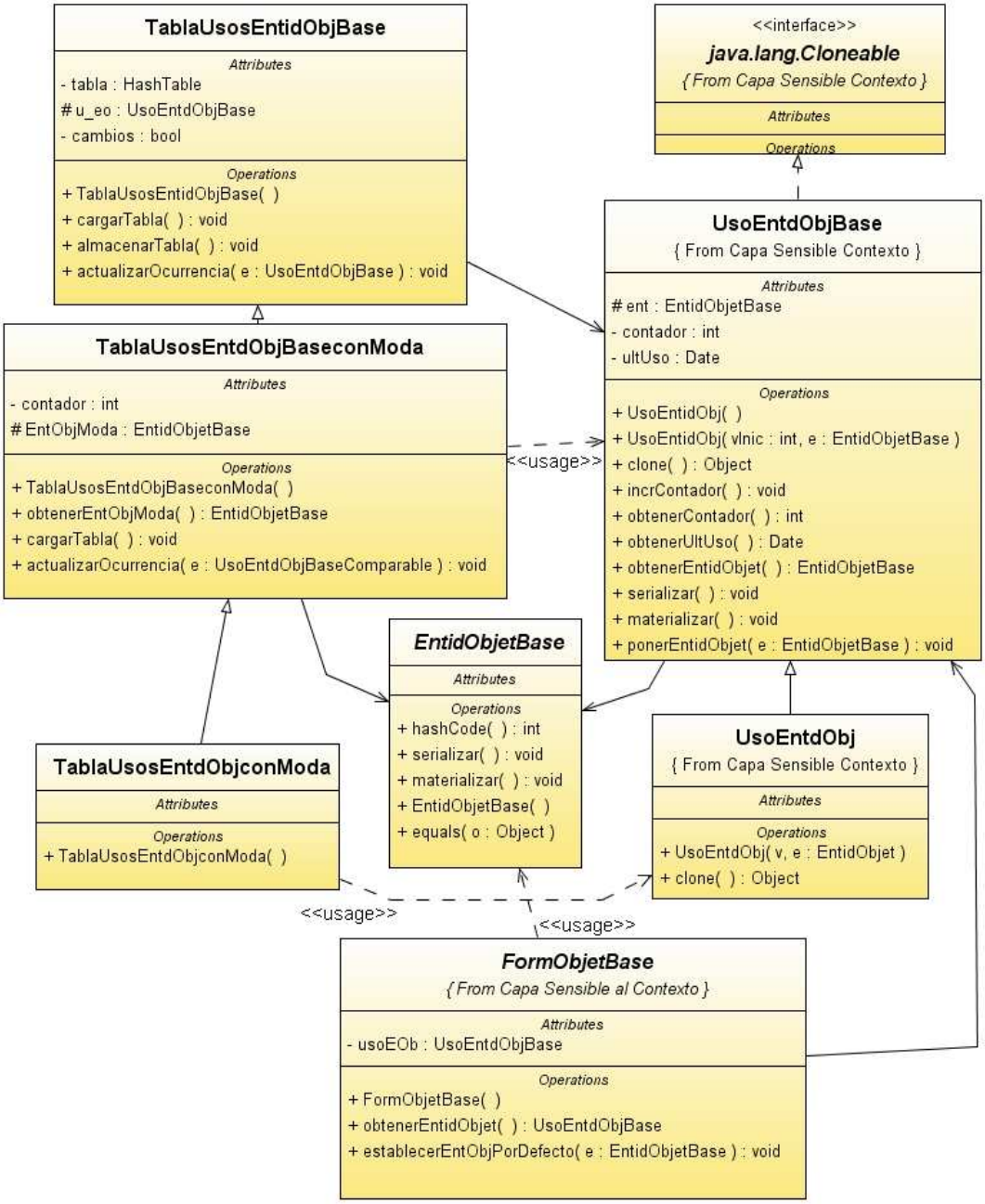


Figura 8.6: Diagrama de clases de la *capa sensible al contexto* para la componente de personalización A en el FPI.

Objetivo de personalización 'ordenación de una lista'

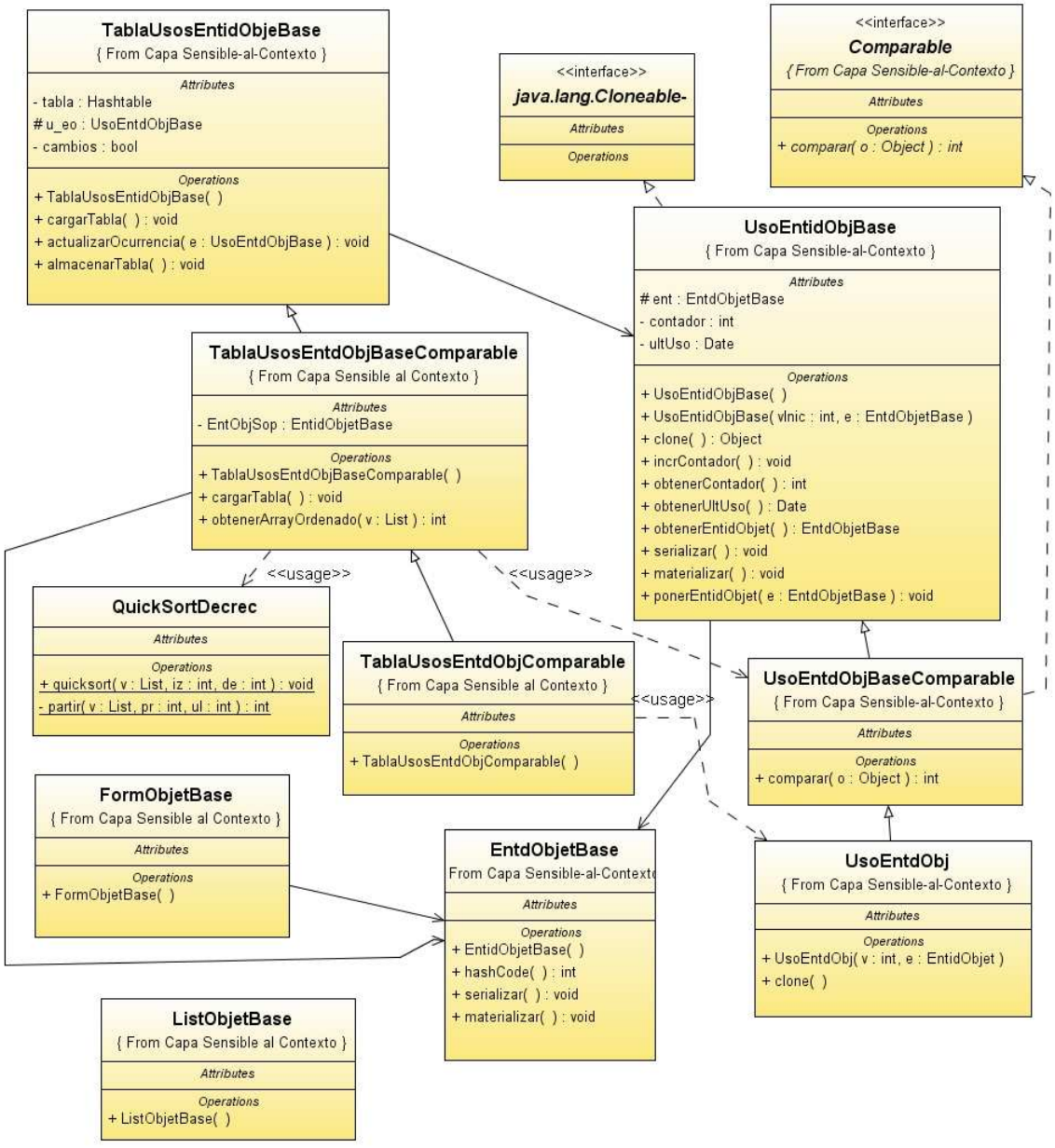


Figura 8.7: Diagrama de clases de la *capa sensible al contexto* para la componente de personalización B para J2ME en el FPI.

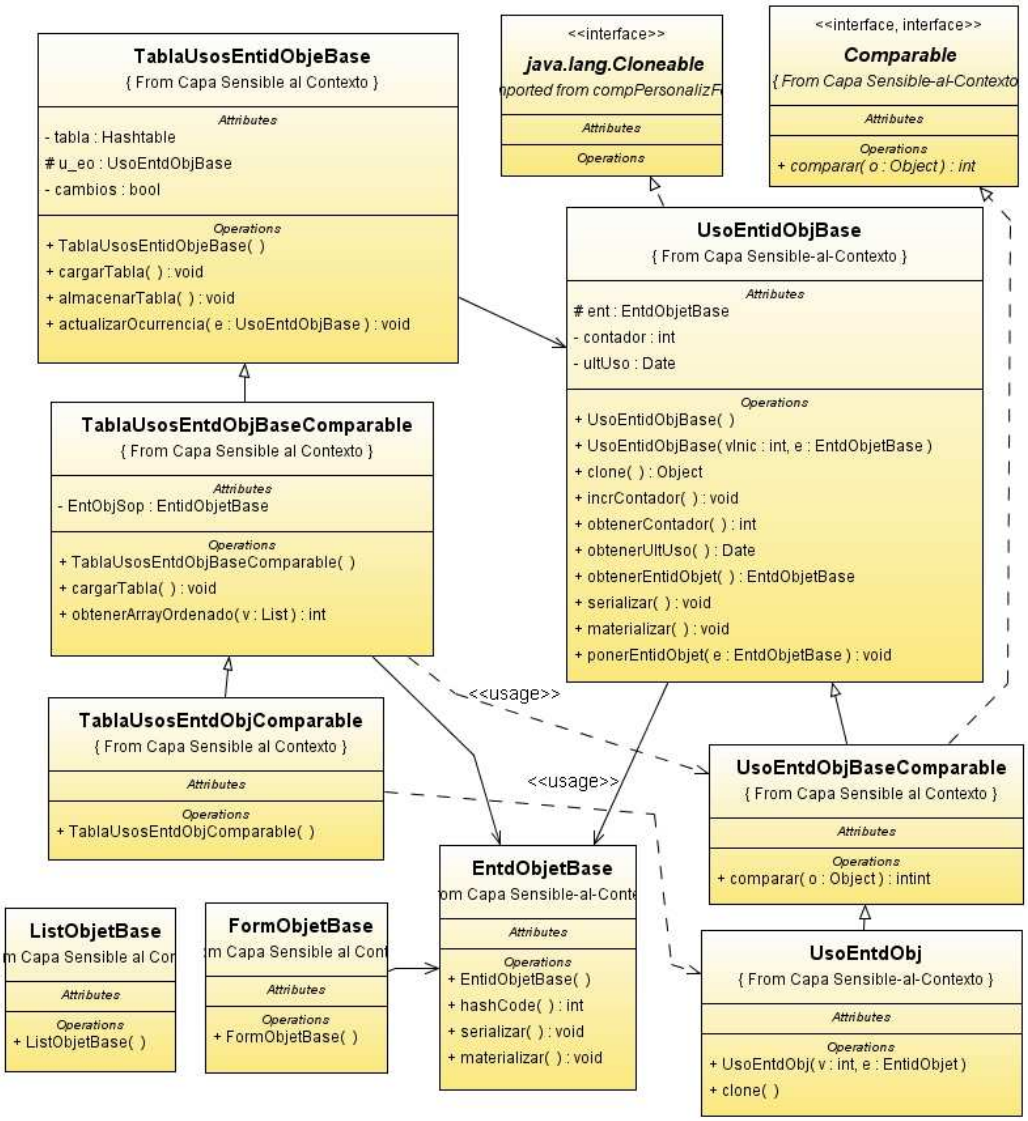


Figura 8.8: Diagrama de clases de la *capa sensible al contexto* para la componente de personalización B para J2SE en el FPI.

Las clases *EntidObjetBase*, *FormObjetBase* y *ListObjetBase* son las clases genéricas expresamente introducidas para aportar genericidad en las referencias a la *entidad objetivo*, el formulario y la lista implicados.

8.2.1.2. Capa aspectual

En la capa aspectual interviene, además del *aspecto CapturadorAdaptador* otro aspecto adicional, denominado *IncrementadorEstructura* en las versiones para aplicaciones

móviles, por ser el *aspecto* encargado de establecer las relaciones de generalización de las nuevas clases genéricas con las clases efectivas (referenciadas en el framework como *EntidObjet*, *FormObjet* y *ListObjet*), así como de incorporar nuevos métodos cuando así se requiere.

En el caso de aplicaciones fijas, en las que se trabaja adicionalmente con *anotaciones de metadatos*, las declaraciones asociadas se incorporan también en este *aspecto* adicional. El nombre designado es en estos casos *AnotadorIncrementador*.

Aspecto CapturadorAdaptador

Se trata del *aspecto* principal que maneja la personalización de una pantalla.

Como las clases del sistema base afectadas por los *aspectos* se integran en una jerarquía donde la super-clase es una clase genérica y las clases de la *capa sensible al contexto* se integran en sus respectivas jerarquías reutilizables, el código de los *aspectos* puede ser escrito utilizando exclusivamente referencias a dichas clases, consiguiendo un código *aspectual* reutilizable. No es hasta el tiempo de ejecución que el código se instancía en las clases efectivas de las *capas lógica* y *sensible al contexto*, consiguiendo que los métodos efectivamente invocados sean los que corresponden a las clases realmente involucradas en cada caso: las *entidades objetivo* manejadas en cada sistema base en particular. Por lo tanto, una misma línea de código admite múltiples comportamientos, a decidir en tiempo de ejecución, mecanismo que se conoce como polimorfismo dinámico.

El polimorfismo dinámico es aplicable en varios puntos del código de los *aspectos*: (1) en los parámetros de los *puntos de corte*; (2) en los códigos de los *consejos* asociados; y (3) en las signaturas de los métodos. Se consigue con ello que todos estos puntos en la definición de los *aspectos* sean independientes del sistema. Aplicando los patrones específicos de *orientación a aspectos*: *template advice*, *abstract pointcut* y *composite pointcut*[6] [HC02] resulta sencillo aislar y desglosar la parte común –independiente del sistema- de la parte dependiente del sistema, estableciendo también una jerarquía de *aspectos* para el *aspecto CapturadorAdaptador*. Así, el *aspecto* base *CapturadorAdaptadorBase* encapsula la parte reutilizable y el *sub-aspecto CapturadorAdaptador* la parte específica.

Objetivo de personalización 'valores por defecto en un formulario'

Para el caso de aplicaciones fijas todos los *puntos de corte* requeridos quedan delegados en *anotaciones de metadatos* (véase *Apéndice B*), no siendo necesario recurrir a ningún patrón.

[6]El *composite pointcut* separa un *punto de corte* en dos o más *puntos de corte* lógicamente independientes, los cuales pueden ser combinados a través de los operadores && y || (véase *Apéndice A*).

En el caso de aplicaciones móviles, de los tres *puntos de corte* requeridos sólo dos se definen como abstractos, a definir por tanto en el *sub-aspecto*. El tercero -el *punto de corte serializarTabla*-, aunque en la figura 8.9 se define particular a la plataforma (perfil MIDP), definiéndolo también como abstracto sería reutilizable para cualquier perfil de J2ME. Otra diferencia respecto a la versión genérica presentada en el *Apéndice B* es que el método constructor del *sub-aspecto CapturadorAdaptador*, a parte de instanciar la tabla requerida en las entidades efectivas a manejar, se encarga también de invocar al método *cargarTabla*, sustituyendo con ello la necesidad del *punto de corte materializarTabla* presentado en la descripción genérica del *Apéndice B*.

El código de los *consejos* es totalmente reutilizable, por lo que se incluye en el *aspecto* base. La aplicación del patrón *abstract pointcut* y *composite pointcut* –utilizado en los *puntos de corte establecerValoresporDefecto* y *capturarCombinación*- en el caso de aplicaciones móviles es suficiente para encapsular las dependencias.

A continuación se muestran los respectivos diagramas de clases para la *capa aspectual* correspondiente a este objetivo de personalización para ambas versiones. En ellos se integran todos los detalles proporcionados en el *Apéndice B*.

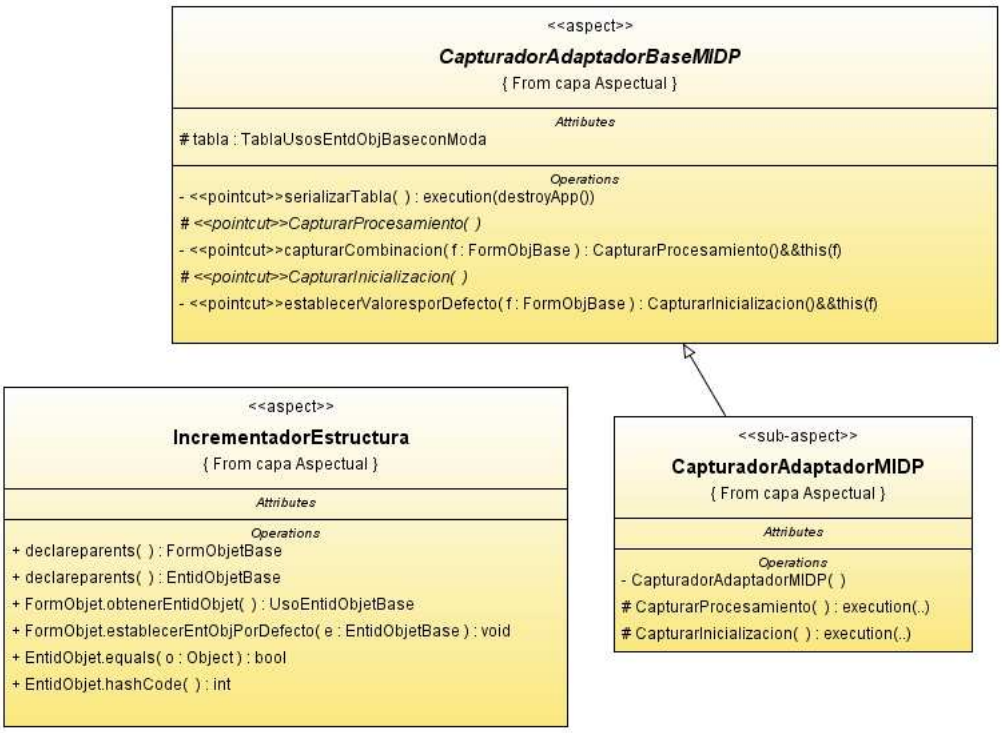


Figura 8.9: Diagrama clases *capa aspectual* componente de personalización A para MIDP (FPI)

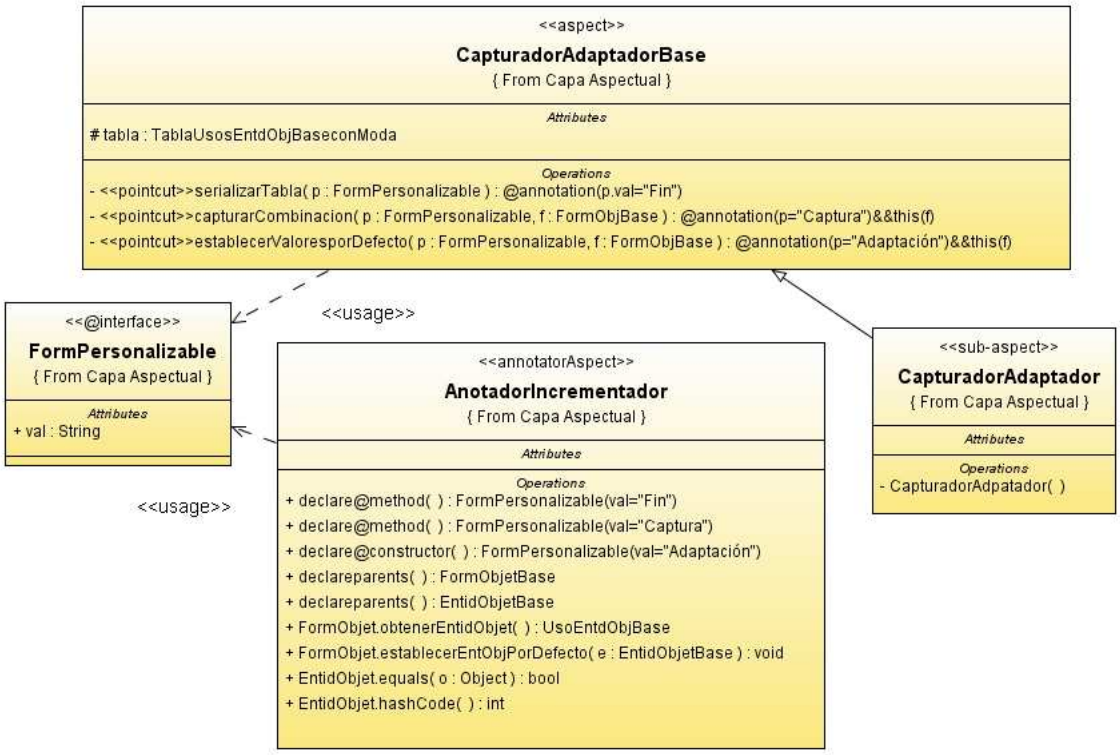


Figura 8.10: Diagrama clases *capa aspectual* componente de personalización A para J2SE (FPI).

FormPersonalizable es un tipo de *anotación* específicamente definido en esta componente para J2SE. Los tres valores para su atributo ("Fin", "Captura" y "Adaptación") permiten diferenciar los distintos *puntos de corte* asociados al código de la aplicación base interceptados por esta componente.

En el *Apéndice C* se presenta el diagrama de clases completo (*capas aspectual* y *sensible al contexto*) para esta componente, donde también queda reflejado cómo afecta a la aplicación base. Pueden consultarse dos diagramas para las dos versiones desarrolladas: J2SE y J2ME.

Objetivo de personalización 'ordenación de una lista'

En la componente para el objetivo de personalización B *('ordenación de una lista')*, en su correspondiente versión para aplicaciones móviles se recurre al patrón *template advice* a fin de completar el código dependiente del sistema en el *sub-aspecto*, además de utilizar los patrones *abstract pointcut* y *composite pointcut* como en la componente anterior. El patrón *template advice* se aplica en tres de los cuatro *puntos de corte* requeridos, definiendo un método abstracto para cada uno (véase figura 8.11 -2 pág. adelante-).

Igual que para el objetivo de personalización anterior, se incorpora un método constructor del *sub-aspecto CapturadorAdaptador*, con el mismo propósito que allí.

Cabe señalar que esta parte es más simple en el caso de aplicaciones fijas, dado que una lista forma parte del propio formulario, en lugar de constituir una pantalla distinta como ocurre con las aplicaciones para móviles, por lo que no se requiere manipular directamente la gestión propia de la lista. Así, el patrón *template advice* sólo se requiere en un *punto de corte*, definiendo en consecuencia un solo método abstracto. El resto de código de los *consejos* es genérico (véase figura 8.12 -2 pág. adelante-).

Un último detalle es que la aplicación del patrón *pointcut composition*[7] permite desglosar aún más las partes específicas en la componente para el perfil MIDP, concretando en mayor medida la parte de código a definir en el *sub-aspecto*.

A continuación se muestran los respectivos diagramas de clases para la *capa aspectual* correspondiente a este objetivo de personalización para ambas versiones. En ellos se integran todos los detalles proporcionados en la *sección 8.1.1*.

[7]Descompone un *punto de corte* complejo en un conjunto de *puntos de corte* componentes, los cuales son lógicamente independientes entre sí.

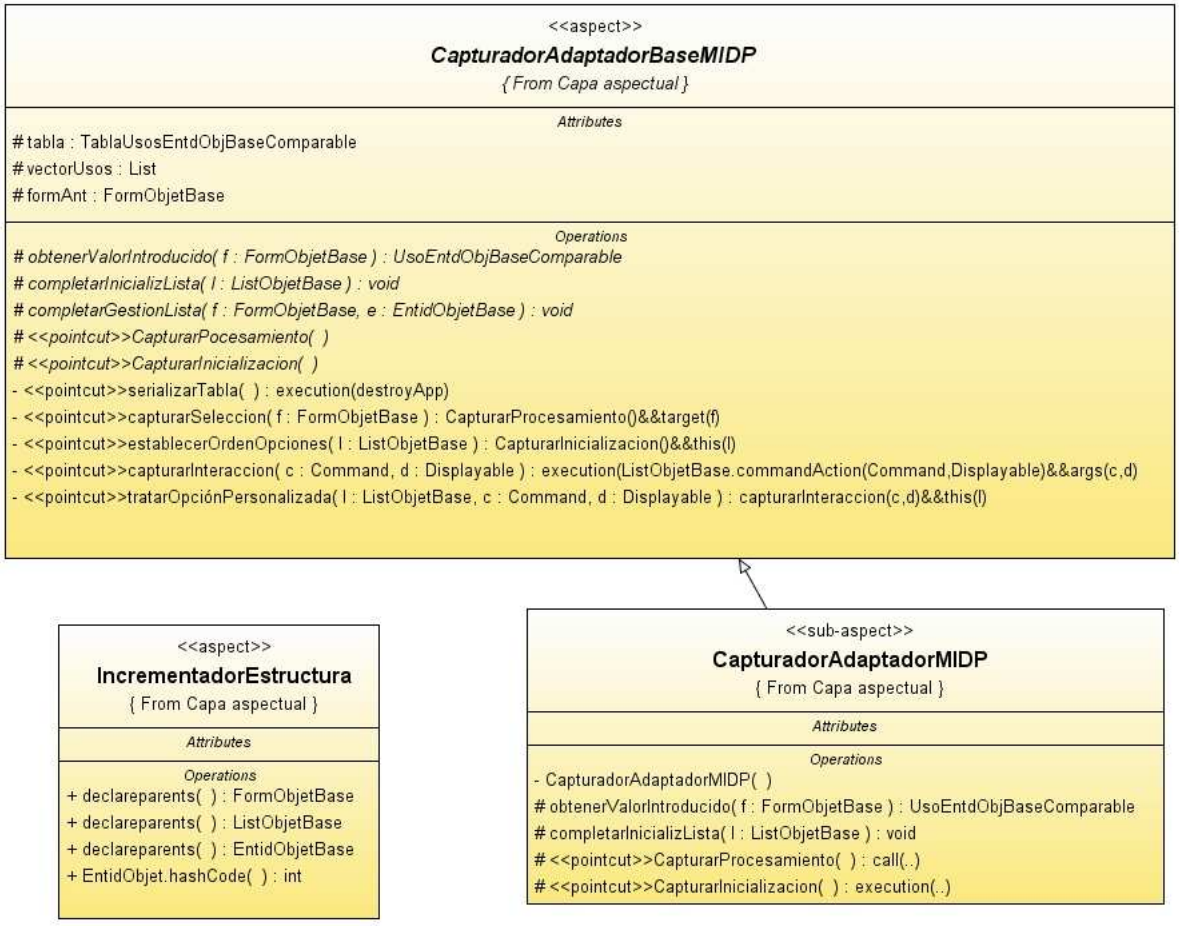


Figura 8.11: Diagrama clases *capa aspectual* componente de personalización B para MIDP (FPI).

ListPersonalizable es un tipo de *anotación* específicamente definido en esta componente para J2SE. Los cuatro valores para su atributo ("Fin", "Captura", "AdaptList" y "AdaptSelecc") permiten diferenciar los distintos *puntos de corte* asociados al código de la aplicación base interceptados por esta componente.

En el *Apéndice C* se presenta el diagrama de clases completo (*capas aspectual* y *sensible al contexto*) para esta componente, donde también queda reflejado cómo afecta a la aplicación base. Pueden consultarse dos diagramas para las dos versiones desarrolladas: J2SE y J2ME.

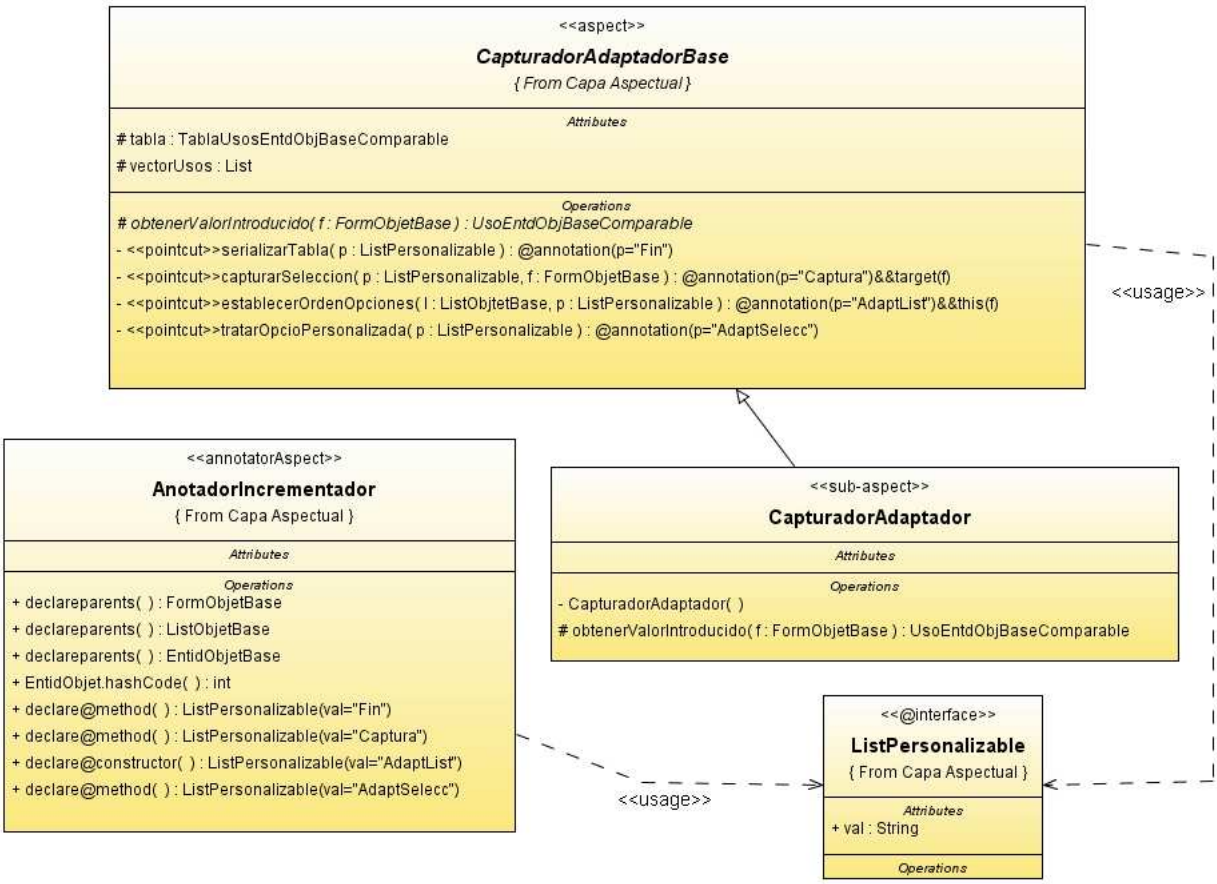


Figura 8.12: Diagrama clases *capa aspectual* componente de personalización B para J2SE (FPI).

8.2.1.3. Consideraciones relativas a la implementación

La instanciación de la componente del framework para la personalización de una pantalla en un sistema concreto requiere completar el código de las siguientes clases y *aspectos*:

Capa sensible al contexto

En la *capa sensible al contexto* se requiere incorporar las clases encargadas de efectuar la instanciación de las clases genéricas en las clases efectivas. Se está haciendo referencia a las clases del nivel inferior para las jerarquías de las *tablas de usos* y la jerarquía de los *usos* (figuras 8.4 y 8.5). Se trata de las clases siguientes:

- la clase hoja de la jerarquía de las tablas: *TablaUsosEntdObjconModa* y *TablaUsosEntdObjComparable* para ambos objetivos de personalización respectivamente.

Para los casos de estudio trabajados ("LectorNoticias" y "ControladoraObras") se trata de las clases *TablaUsosPerfilconModa* y *TablaUsosInformeconModa*, respectivamente para el objetivo de personalización *'valores por defecto de un formulario'*.

Para el objetivo de personalización *'ordenación de una lista'* se trata de las clases *TablaUsosContactoComparable* y *TablaUsosMaterialComparable*, respectivamente para los dos sistemas.

- la clase hoja de la jerarquía de los usos, la cual aparece referenciada en el framework como *UsoEntdObj*.

  Para los dos casos de estudio trabajados, y concretamente para el objetivo de personalización *'valores por defecto de un formulario'* se trata de las clases *UsoPerfil* y *UsoInforme* las cuales descienden directamente de la clase *UsoEntidObjBase*. Para el objetivo de personalización *'ordenación de una lista'* se trata de las clases *UsoContacto* y *UsoMaterial*, para uno y otro sistema, las cuales descienden de la clase *UsoEntdObjBaseComparable*.

Capa aspectual

En la *capa aspectual* se requieren completar los siguientes aspectos:

- el *sub-aspecto CapturadorAdaptador –CapturadorAdaptadorMIDP* para aplicaciones móviles-, el cual en el objetivo de personalización *'valores por defecto de un formulario'* tan sólo debe concretar los *puntos de unión* a interceptar en el sistema base, si se trata de una aplicación móvil. Si se trata de aplicaciones fijas se deja delegado en *anotaciones de metadatos*.

  En el objetivo de personalización *'ordenación de una lista'*, además, deben implementarse los métodos declarados como abstractos en el *aspecto* base.

  En cualquier caso, la constructora del método resulta imprescindible. Recordemos que es la responsable de instanciar la tabla de usos en la entidad objetivo propia del sistema.

- el *aspecto* adicional, denominado *IncrementadorEstructura* para aplicaciones móviles y *AnotadorIncrementador* para aplicaciones fijas.

8.2.2. Generalización de la componente de personalización de una funcionalidad de la aplicación base

Igual que para la componente genérica de personalización de una pantalla, el framework proporciona dos versiones para esta componente: la versión para aplicaciones móviles en

J2ME y la versión basada en J2SE, donde la principal diferencia es la utilización o no de *anotaciones de metadatos*.

8.2.2.1. Capa sensible al contexto

La estructura de esta capa ha quedado explicitada en el *Apéndice B*. Tal y como allí se expone, toda la funcionalidad recae sobre la clase *GestiónParmFuncionalidadExistente*, donde se establece una separación entre los métodos comunes, que se definen en la propia clase, y los que contienen las particularidades de cada caso particular. Éstos últimos se declaran a través de la interfaz *comparadorParam*, a efectos de garantizar su signatura e identificar la parte de código que debe ser definida cada vez que se desea instanciar esta componente en un sistema concreto. Esta capa ofrece estas dos variaciones respecto a la versión genérica presentada en el *Apéndice B*:

1. Se introduce una clase genérica para hacer referencia a la *entidad objetivo* manejada. Su nombre es el mismo utilizado en los componentes anteriores: *EntidObjetBase*.

2. Dado que el mecanismo de almacenamiento persistente para J2SE es muy distinto al de J2ME, basado en colecciones de registros (los llamados *RecordStore*), se cree oportuno establecer una jerarquía de gestores para aislar y diferenciar el código que maneja el almacenamiento persistente (métodos *resgistrarParametros*, *serializar* y *materializar*). Es por ello que el framework ofrece en total tres clases gestoras. En el nivel superior –la clase *GestionParmFuncionlExistBase*- aparecen los métodos que no trabajan este aspecto, y en las sub-clases *GestionParmFuncionlExist* y *GestionParmFuncionlExistJ2ME* se incluyen los métodos mencionados, utilizando *DataStreams* y *RecordStores* respectivamente.

Otra observación que, a pesar de no haber sido plasmada en el framework, conviene tener en cuenta de cara a aumentar su reutilización, es el tipo del parámetro a ser personalizado. Como ya se comenta en el *Apéndice B*, se ha presupuesto de tipo *long*, aunque en algunas ocasiones podría tratarse de un tipo real. Con objeto de incorporar también esta variabilidad en el framework, bastaría con definir los atributos *parametroAPersonalizar* y *valorDuranteMuestreo*, así como el método *obtenerParametroAPersonalizar* en las sub-clases, en lugar de hacerlo en la clase base. Así, podría definirse una clase específica para cada posible tipo de parámetro. No obstante, como el tipo real no está soportado en J2ME, esta última variación en la jerarquía tan sólo se aplicaría sobre la rama de gestores para aplicaciones fijas.

El diagrama de clases resultante de aplicar estas consideraciones queda reflejado en la figura 8.13. En él se integran todos los detalles proporcionados en el *Apéndice B*.

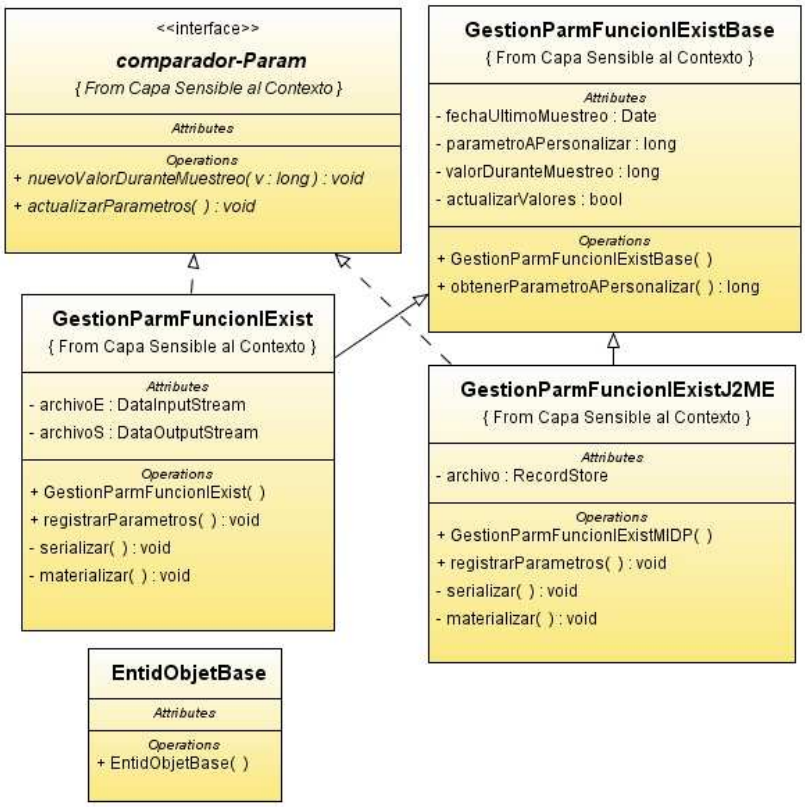


Figura 8.13: Diagrama clases *capa sensible al contexto* componente de personalización de una funcionalidad para J2SE y J2ME (FPI).

8.2.2.2. Capa aspectual

En la *capa aspectual* intervienen, además del *aspecto IncrementadorEstructura – AnotadorIncrementador* en aplicaciones fijas-, el *aspecto CapturadorAdaptador*.

Al igual que en los componentes de personalización de una pantalla presentados anteriormente, se recurre a una pequeña jerarquía de *aspectos* con el mismo propósito. Así, se define el *aspecto CapturadorAdaptadorBaseJ2ME* como super-clase y el *aspecto CapturadorAdaptadorJ2ME* como sub-clase para aplicaciones móviles, y el *aspecto CapturadorAdaptadorBase* y *CapturadorAdaptador*, respectivamente para aplicaciones fijas. La aplicación del patrón *template advice* en el *punto de corte CapturarComportam* da lugar a la definición del método abstracto *obtenerValorMuestreo*.

En la versión para aplicaciones móviles, al no poder recurrir al uso de metadatos, se aplica el patrón *abstract pointcut* para los tres *puntos de corte*, pudiendo incorporar el *aspecto* base igualmente el código completo de los *consejos* asociados, excepto para el caso

del *punto de corte CapturarComportam*, donde, al igual que en la versión para aplicaciones fijas, se aplica el patrón *template advice* definiendo el método *obtenerValorMuestreo* como abstracto.

A continuación se muestran los diagramas de clases correspondientes a esta capa para las dos versiones. En ellos se integran todos los detalles proporcionados en el *Apéndice B*.

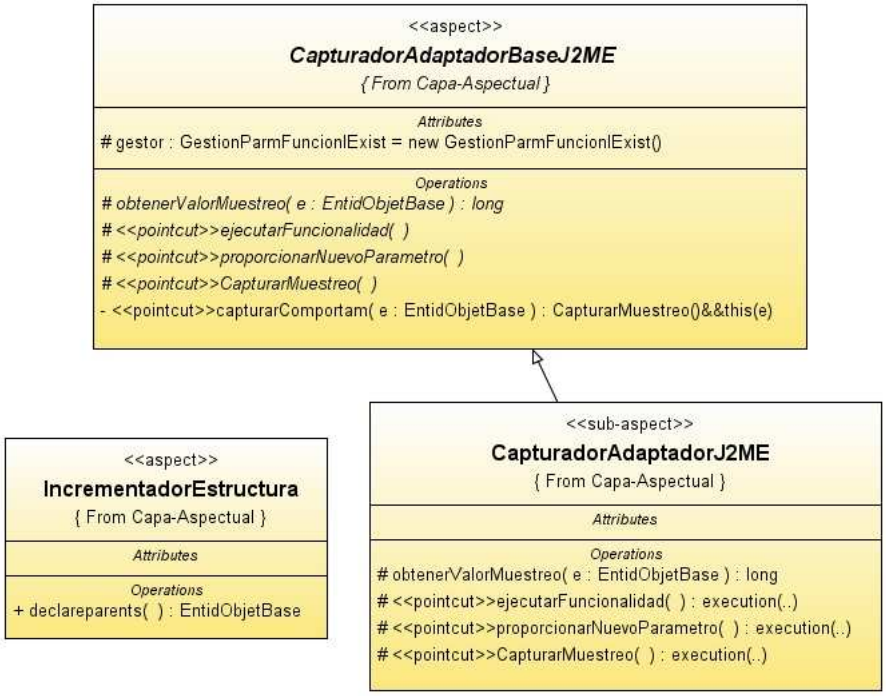


Figura 8.14: Diagrama clases *capa aspectual* componente de personalización de una funcionalidad para J2ME (FPI).

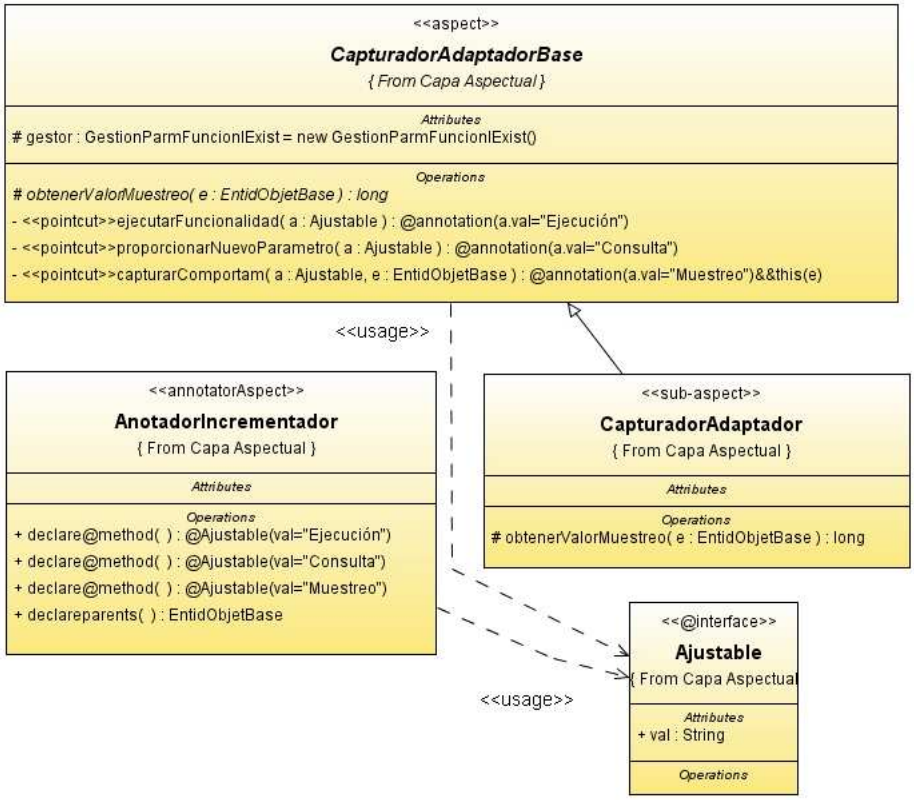


Figura 8.15: Diagrama clases *capa aspectual* componente de personalización de una funcionalidad para J2SE (FPI).

Ajustable es un tipo de *anotación* específicamente definido en esta componente para J2SE. Los tres valores para su atributo ("Ejecución", "Consulta" y "Muestreo") permiten diferenciar los distintos *puntos de corte* asociados al código de la aplicación base interceptados por esta componente.

En el *Apéndice C* se presenta el diagrama de clases completo (*capas aspectual* y *sensible al contexto*) para esta componente, donde también queda reflejado cómo afecta a la aplicación base. Pueden consultarse dos diagramas para las dos versiones desarrolladas: J2SE y J2ME.

8.2.2.3. Consideraciones relativas a la implementación

La instanciación de esta componente del framework en un sistema concreto requiere la completitud de las siguientes clases y *aspectos*:

- En la *capa sensible al contexto* los métodos declarados en la interfaz *comparadorParam*, los cuales son específicos del sistema.
- el *sub-aspecto CapturadorAdaptador* para aplicaciones fijas, o bien *CapturadorAdaptadorJ2ME* para aplicaciones móviles, el cual debe implementar el método *obtenerValorMuestreo*, y también establecer los *puntos de corte* en este último tipo de aplicaciones, definidos como abstractos en el *aspecto* base.
- y por supuesto, el *aspecto* adicional *IncrementadorEstructura* o *AnotadorIncrementador*, dependiendo de si se trata de aplicaciones móviles o de aplicaciones fijas.

8.2.3. Generalización de la componente de apoyo al trabajo en grupo

El framework proporciona también dos versiones para esta componente: la versión para aplicaciones móviles en J2ME y la versión basada en J2SE.

8.2.3.1. Capa sensible al contexto

El FPI completa la versión genérica descrita en el *Apéndice B* incorporando un par de clases que representan las clases genéricas de dos *entidades objetivo* que suelen estar presentes en actividades colaborativas: la entidad tarea y la entidad material. Tal y como se observa en la figura 8.16 (2 pág. adelante), reciben el nombre de *TareaBase* y *MaterialBase*.

Existen otras entidades conceptuales que, a pesar de que no hayan sido requeridas en el caso de estudio trabajado, son susceptibles de aparecer en un entorno colaborativo, como por ejemplo la entidad rol o actor. En previsión de ello podrían ser incorporadas también en el framework sus correspondientes clases genéricas. En el momento de instanciarlo en una aplicación concreta se identificarían cuáles de ellas son manejadas y cuáles no, estableciendo las correspondientes dependencias con el sistema.

La clase *cGrupoParticular* representa la *consciencia de grupo particular* (véase *definición* en *Capítulo 2*; *sección 2.1.3.*), una pieza de código ideada para recoger la percepción que cada individuo tiene de la actividad colaborativa, con el propósito de aportar la información necesaria para construir un *conocimiento compartido* en el *servidor de plasticidad*. Los atributos propuestos para esta clase están sujetos a variación, tanto por exceso como

por defecto. Sin embargo, algunos de ellos son de uso universal. Los atributos que se consideran comunes a cualquier sistema son los siguientes: *estado* y *tareasPendientes*. En el caso en que se lleve un control respecto a las tareas canceladas aparecerían también los atributos *tareasCanceladas* y *notifTareasCanc*. Aunque aquí tan sólo se muestra una clase *cGrupoParticular*, fácilmente podría construirse una jerarquía de clases para establecer categorías y favorecer la reutilización de esta unidad de programa.

La clase *informcPropia* es útil para la identificación de los miembros del grupo, a fin de conocer qué individuos han realizado ciertas operaciones de interés para el grupo. Entre los atributos propuestos, concretamente el del rol, resulta especialmente interesante puesto que plantea la posibilidad de restringir acciones a roles específicos, e incluso la posibilidad de variar el rol de un miembro del grupo a lo largo de la actividad.

En cuanto a la clase *informcEquipo*, es también factible de ser utilizada en diversos sistemas colaborativos, en concreto siempre y cuando la actividad colaborativa esté repartida entre distintos equipos, concebidos como subgrupos del grupo de trabajo. En estos casos disponer de la información de contacto con los miembros del equipo resulta interesante, tal y como ocurre en el caso de estudio trabajado. La clase *informcEquipo* corresponde a una versión genérica de la clase *EquipoTarea* que aparece en los diseños de los casos de estudio del *Capítulo 7*.

A continuación se muestra el diagrama de clases correspondiente a esta capa, que en este caso coincide para las dos versiones. En él se integran todos los detalles proporcionados en el *Apéndice B*.

8.2.3.2. Capa aspectual

Al ser tres las unidades de programa *aspecto* en que se ha dividido el tratamiento a llevar a cabo en esta componente (*Comunicación*, *Colaboración* y *Coordinación*), el framework proporciona un *aspecto* abstracto y uno concreto para cada uno de ellos, reuniendo un total de seis unidades *aspecto*, tal y como se muestra en los diagramas de clases de las figuras 8.17 y 8.18 (2 pág. adelante). Por supuesto, también interviene el *aspecto IncrementadorEstructura*.

A continuación se muestran los diagramas de clases correspondientes a esta capa para las dos versiones. En ellos se integran todos los detalles proporcionados en el *Apéndice B*.

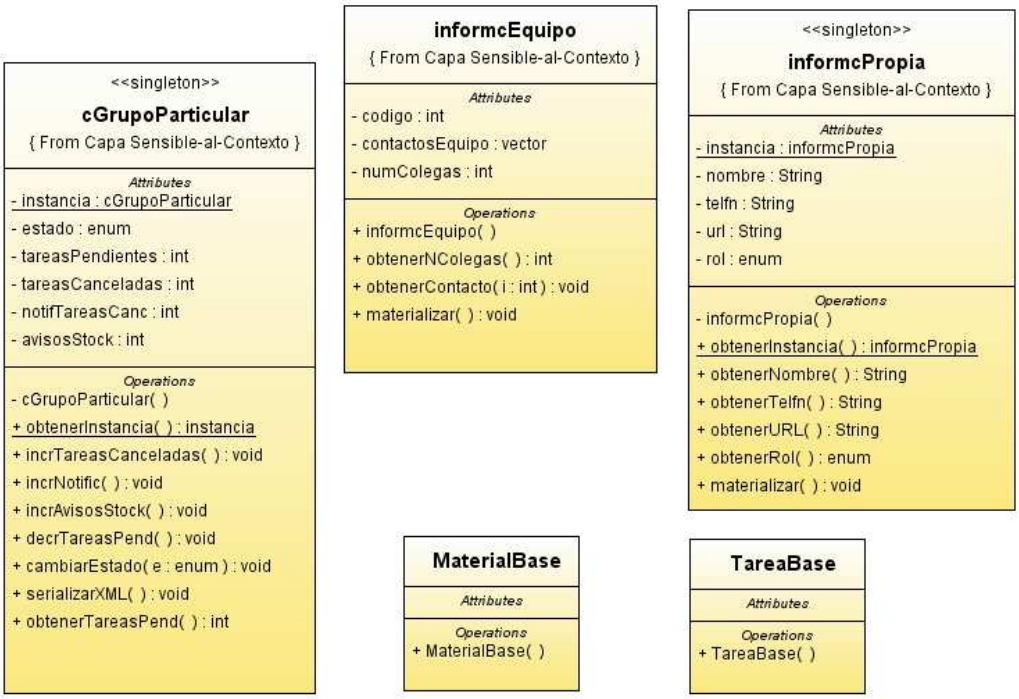


Figura 8.16: Diagrama clases *capa sensible al contexto* componente de apoyo al trabajo en grupo (FPI).

SensibleGrupo es un tipo de *anotación* específicamente definido en esta componente para J2SE. Los tres valores para su atributo ("Comunicación", "Coordinación" y "Colaboración") permiten diferenciar los distintos *puntos de corte* asociados al código de la aplicación base, interceptados por esta componente.

En el *Apéndice C* se presenta el diagrama de clases completo (*capas aspectual* y *sensible al contexto*) para esta componente, donde también queda reflejado cómo afecta a la aplicación base. Pueden consultarse dos diagramas para las dos versiones desarrolladas: J2SE y J2ME.

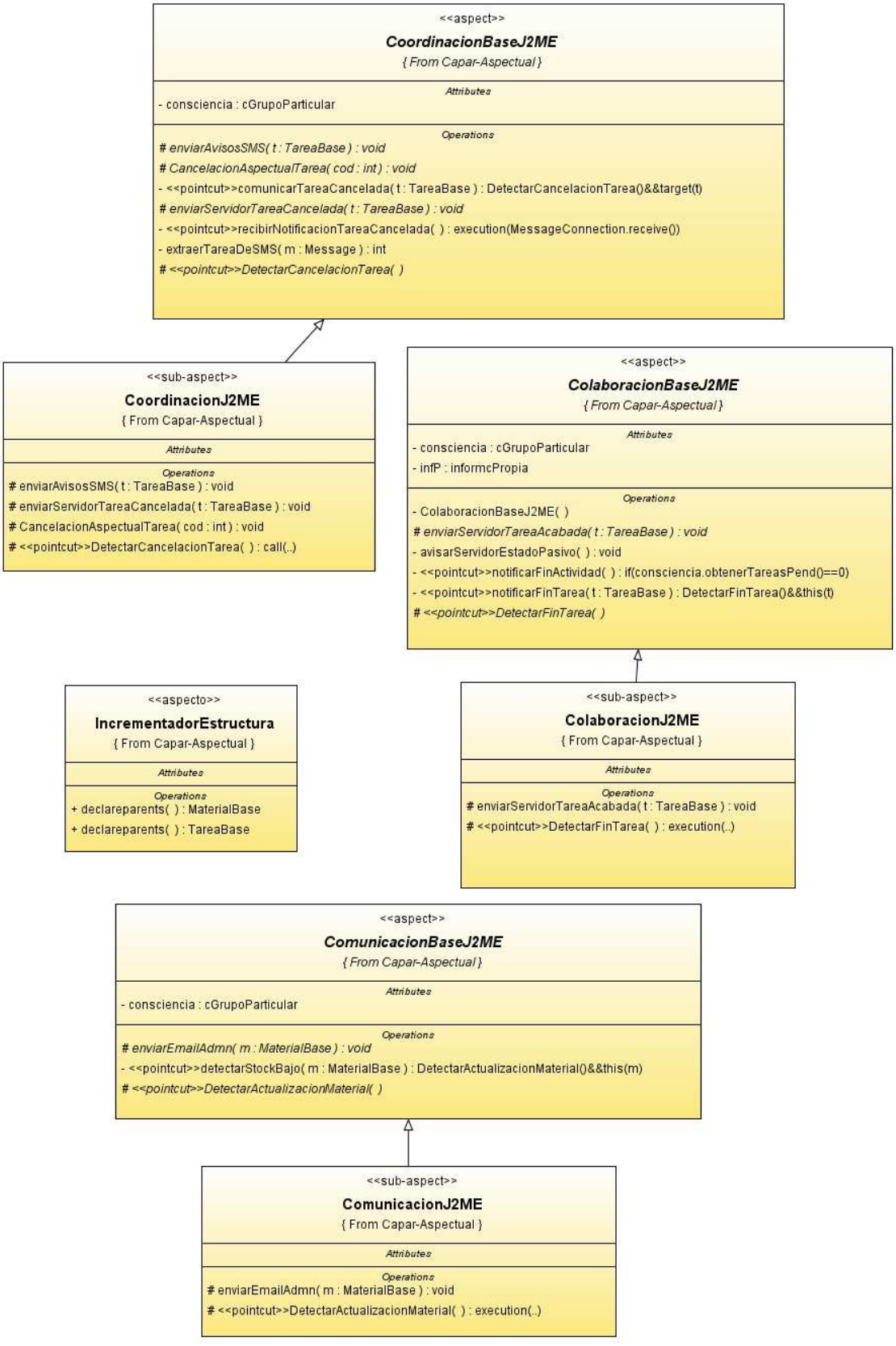


Figura 8.17: Diagrama clases *capa aspectual* componente apoyo trabajo en grupo para J2ME (FPI).

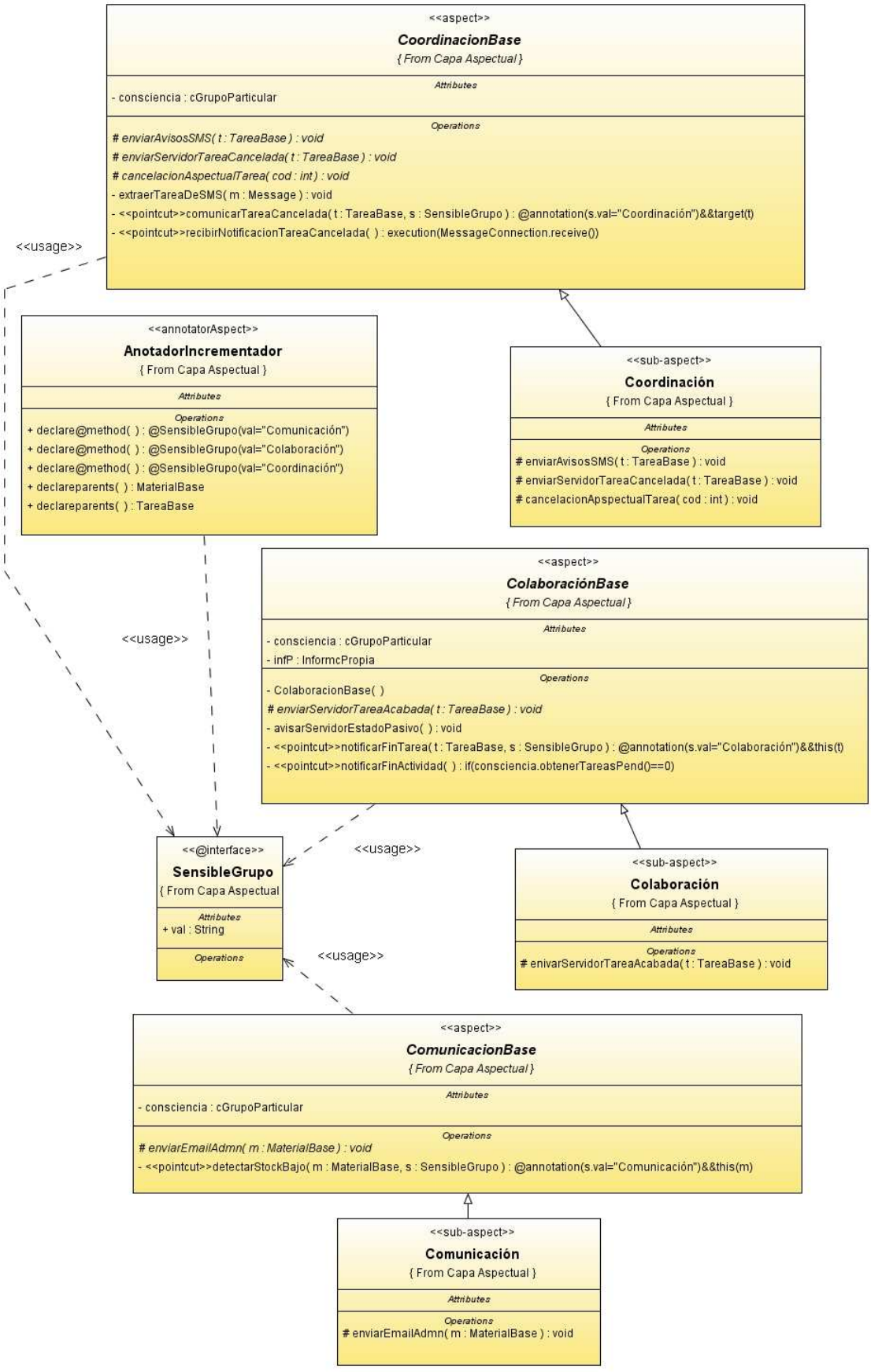


Figura 8.18: Diagrama clases *capa aspectual* componente apoyo trabajo en grupo para J2SE (FPI).

8.2.3.3. Consideraciones relativas a la implementación

Según todo lo expuesto, la instanciación del framework en un sistema concreto consiste únicamente en completar los tres *sub-aspectos* con el código específico del sistema base. De hecho, dada la capacidad de reutilización que proporciona el uso de clases genéricas, tan sólo queda pendiente definir los métodos abstractos en el caso de aplicaciones en J2SE. Para aplicaciones móviles se requiere definir tanto los métodos abstractos como los *puntos de corte* abstractos. En este último caso los *puntos de corte* a concretar son muy puntuales, dado que la aplicación del patrón *composite pointcut* permite delegar únicamente en el *sub-aspecto* una parte concreta de la definición del *punto de corte*. El código correspondiente a los *consejos* asociados a cada *punto de corte* no requiere ser redefinido.

Por último, tal y como ya se ha comentado, en la *capa sensible al contexto* es posible que se requieran realizar ciertas variaciones en las clases propuestas aquí. Si ir más lejos, la clase *cGrupoParticular* puede requerir ser ampliada con nuevos atributos y métodos. Por otro lado, la clase *informcEquipo* en ocasiones puede resultar innecesaria, o incluso es posible encontrarse con otro tipo de necesidades que no han sido reflejadas aquí. En ese caso se ampliaría o sustituiría dicha clase por la que se considere más apropiada.

8.2.4. Generalización de la componente de adaptación al entorno o restricción hardware

Tal y como ya ha sido comentado, esta es la componente que por las razones ya expuestas –básicamente que los parámetros que determinan la adaptación no proceden del uso o ejecución de la aplicación-, se caracteriza por su alto grado de ortogonalidad respecto al sistema base. Este desacoplamiento se traduce en que generalmente esta funcionalidad extra puede ser incorporada sin incurrir en ninguna dependencia con las clases del código subyacente. En consecuencia, no es necesario recurrir a la construcción de una superclase genérica como abstracción de las *entidades objetivo* manipuladas en el sistema, tal y como ha sido necesario en las demás componentes.

Aunque en principio este tipo de tratamientos tan sólo resulta útil para aplicaciones móviles, se han diseñado igualmente dos versiones pensando en los ordenadores portátiles, los cuales se programan con la modalidad J2SE.

8.2.4.1. Capa sensible al contexto

Tal y como se expone en el *Capítulo 7* (véase *Capítulo 7 - sección 7.3.1.2.4.*), esta componente es suficientemente genérica como para resolver el tratamiento de cualquier tipo

de factor relacionado con el entorno o restricción hardware. La parte específica, tanto de captura como de tratamiento de cada factor se encuentran convenientemente encapsuladas, evitando que esas variaciones afecten en el resto del código de la componente. Esto le otorga un alto grado de reutilización, a la vez que le imprime la propiedad de independencia de la plataforma.

Como consecuencia de la ortogonalidad de esta capa con respecto a la aplicación subyacente, no es necesario introducir ninguna variación en el diseño respecto a la versión genérica presentada en el *Apéndice B*. La figura 8.19 (pág. siguiente) muestra el diagrama de clases. Tan sólo se producen ligeras variaciones en los nombres de las unidades de programa: *GestorFactor* en lugar de *GestorFactorExterno* y *perceptorEntnOHardw* en lugar de *perceptorEntorno*, a fin de introducir una connotación más generalizada a ambos tipos de atributos del contexto.

A continuación se muestra el diagrama de clases correspondiente a esta capa, que es común para las dos versiones. En él se integran todos los detalles proporcionados en el *Apéndice B*.

8.2.4.2. Capa aspectual

Tal y como se ha visto en los dos casos de estudio expuestos, la frecuencia y las situaciones en las que es conveniente analizar el estado del factor objeto de tratamiento varían según las necesidades de cada caso. Recordemos que esta componente pretende abarcar el tratamiento de un amplio abanico de *restricciones de tiempo real*. Con el propósito de dar respuesta a diversas necesidades, la jerarquía de *aspectos* de esta componente es algo más compleja que en los componentes anteriores. Con objeto de facilitar la programación de tratamientos tan diversos se distinguen tres niveles de interactividad con el sensor, los cuales se modelan a través de tres *sub-aspectos* distintos, tal y como aparece en las figuras 8.20 y 8.21 (2 pág. adelante). De izquierda a derecha aparecen ordenados de menor a mayor intensidad de comunicación con el sensor (*ContFEntrnOHardBajoDemanda*, *ContFEntrnOHardInteractivo* y *ContFEntrnOHardBackground* para J2SE y *ContrFEntrnHardBajDemMIDP*, *ContrFEntrnOHardInteracDemMIDP* y *ContrFEntrnOHardBackgMIDP* para el perfil MIDP).

El *punto de corte ActivarConsultaSensor* es común a todos ellos. Se ocupa de activar la comunicación con el sensor, con objeto de obtener una nueva lectura correspondiente al factor objeto de tratamiento.

El *punto de corte cerrarHistórico* también es común para todos los casos. Se ejecuta al finalizar la sesión con el sistema, a efectos de recoger un histórico. Cabe recordar que

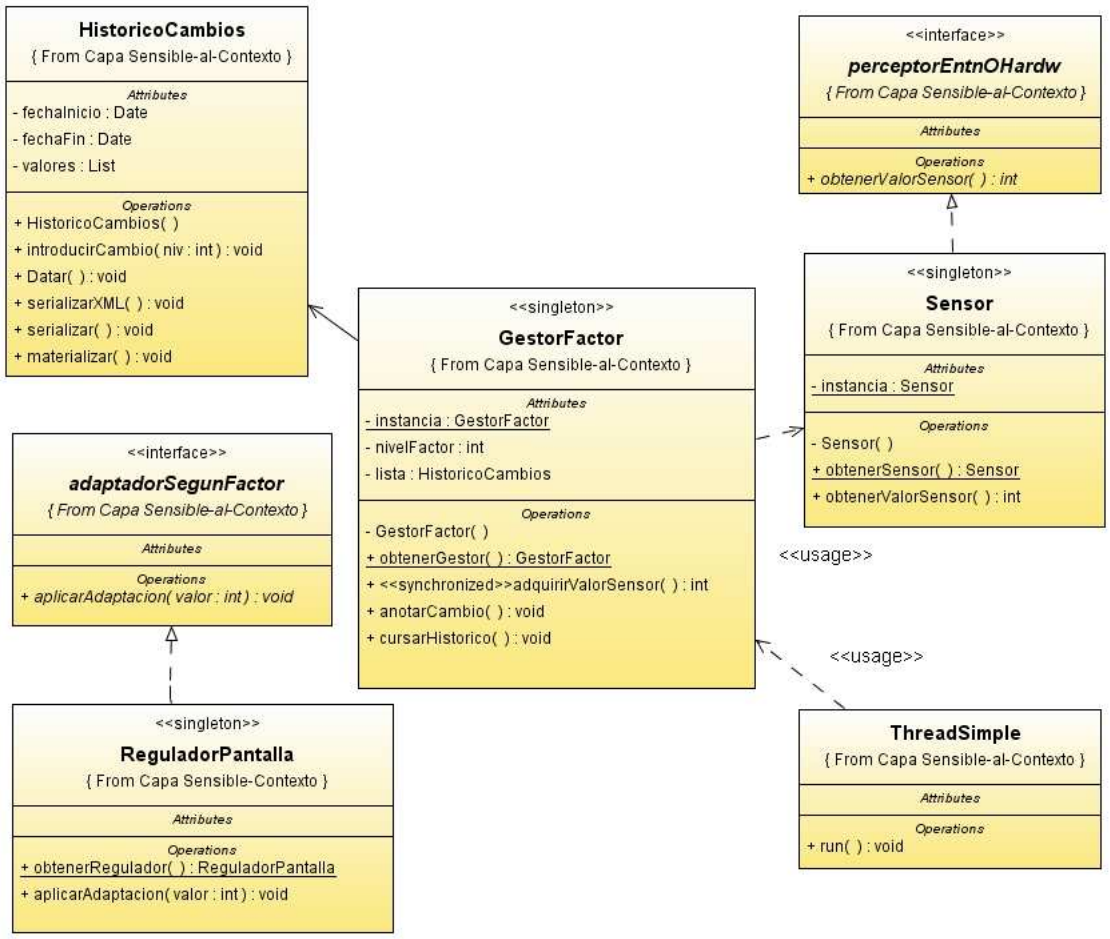


Figura 8.19: Diagrama clases *capa sensible al contexto* componente adaptación entorno o hardware (FPI).

éste es uno de los requisitos identificados por Dey en la construcción de infraestructuras del contexto [Dey00]. Ambos *puntos de corte* aparecen en el *aspecto* base.

A continuación se muestran los diagramas de clases correspondientes a esta capa para las dos versiones (J2SE y el perfil MIDP). En ellos se integran todos los detalles proporcionados en el *Apéndice B*.

SensibleContexto es un tipo de *anotación* específicamente definido en esta componente para J2SE. Los dos valores para su atributo ("Consulta" y "Fin") permiten diferenciar los distintos *puntos de corte* asociados al código de la aplicación base interceptados por esta componente. En este caso, el *aspecto* adicional recibe el nombre de "Anotador", a diferencia de las otras componentes donde se denomina "AnotadorIncrementador". La razón de este

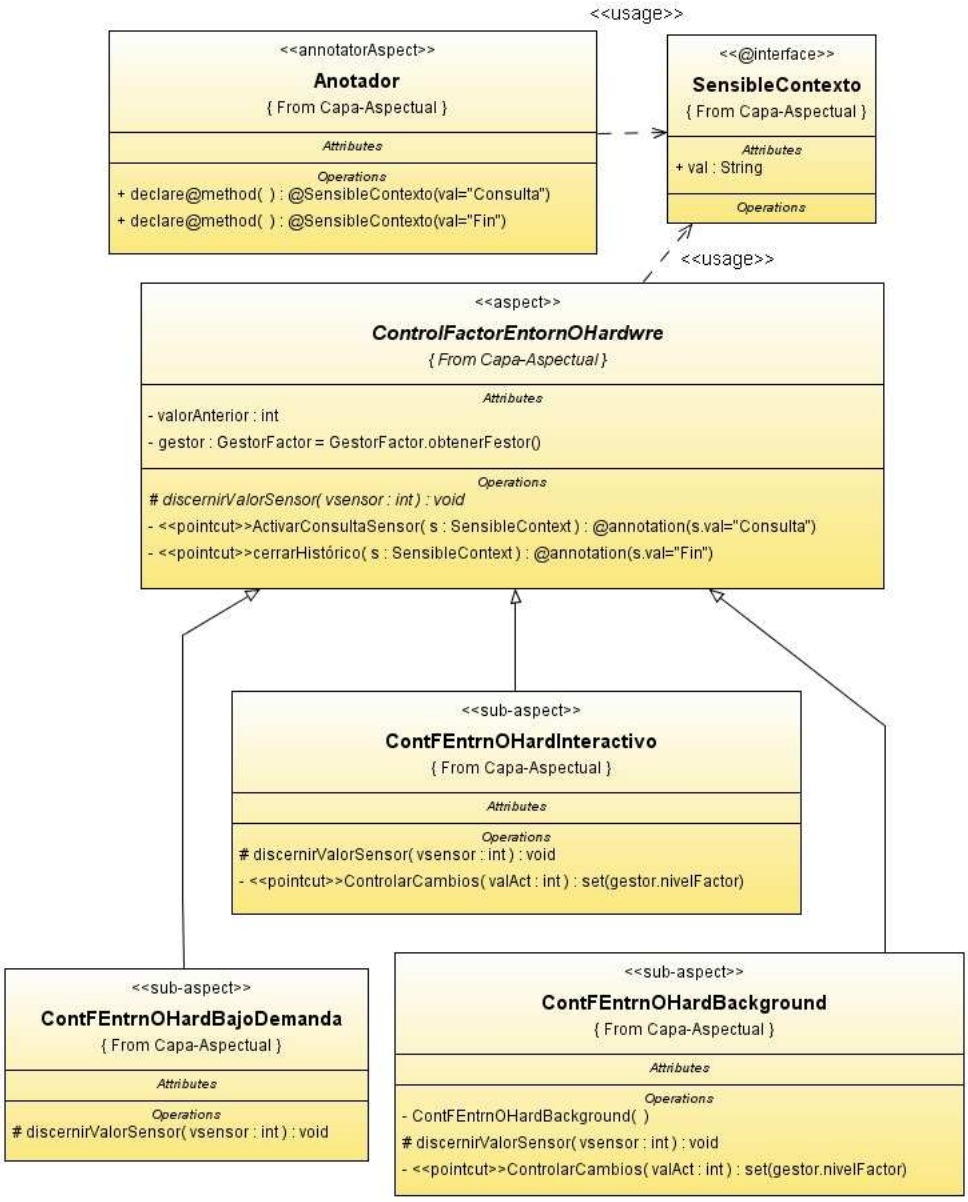


Figura 8.20: Diagrama clases *capa aspectual* componente adaptación entorno o hardware para J2SE (FPI).

cambio es que en esta componente no es necesario introducir ninguna declaración intertipo, puesto que no se requiere alterar la estructura estática del programa base.

A continuación se comentan las diferencias entre los distintos *sub-aspectos* que intervienen en esta componente del framework.

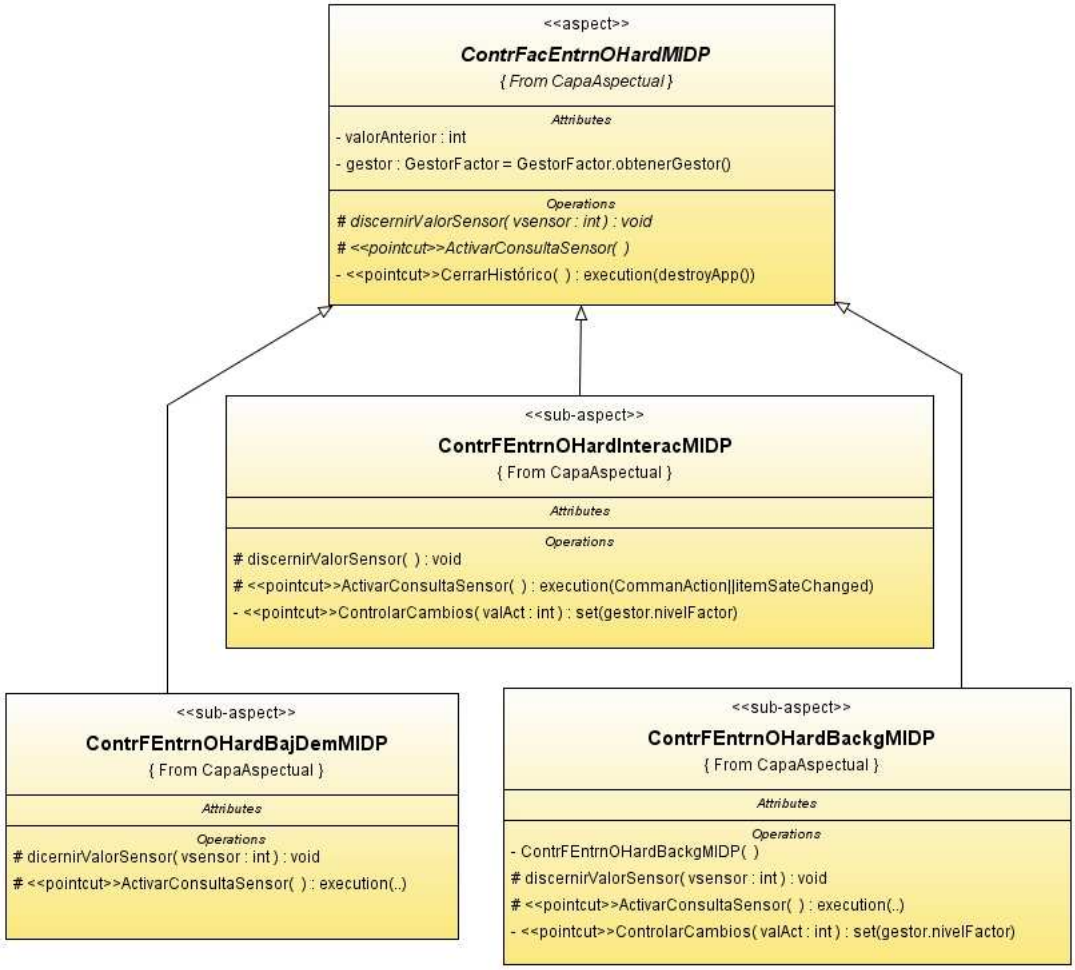


Figura 8.21: Diagrama clases *capa aspectual* componente adaptación entorno o hardware para MIDP (FPI).

Aspecto ContFEntrnOHardBajoDemanda

El *aspecto ContFEntrnOHardBajoDemanda* representa aquellos casos donde la necesidad de comunicación con el sensor es puntual, y por tanto localizada en puntos muy concretos de la ejecución del sistema base, aquellos que son interceptados a través del punto de corte *ActivarConsultaSensor*. Tanto la comunicación con el sensor como el tratamiento correspondiente se incluyen en el *consejo* asociado a dicho *punto de corte*. En concreto, toda la responsabilidad acerca del tratamiento, análisis y toma de decisiones al respecto del factor objeto de tratamiento recae sobre el método *discernirValorSensor*, a especializar por este *aspecto*, como resultado de aplicar el patrón *template advice*. Parte de la operativa correspondiente a *discernirValorSensor* puede delegarse en el método estático

aplicarAdaptacion implementado por la clase *ReguladorPantalla*.

Este *aspecto*, que concentra todo el tratamiento en un sólo *punto de corte* (*ActivarConsultaSensor*), es el candidato apropiado para el tratamiento de las restricciones en recursos hardware (e. g. nivel de batería, memoria disponible o nivel de conectividad en red), donde la acción de adaptación a aplicar se localiza en funcionalidades concretas del sistema base (e. g. en función de la memoria disponible en el momento de llevar a cabo una cierta operación que comporta la visualización de gran cantidad de información, combinar o no la información textual a procesar con información gráfica o multimedia). También sería apropiado para ciertos factores ambientales cuyo tratamiento está también localizado en ciertas operaciones, como pudiera ser el factor posición del usuario, a tener en cuenta, por ejemplo, al refrescar la posición relativa del usuario en un mapa de la zona. Dependiendo del nivel de acoplamiento con la aplicación requerido para llevar a cabo esta funcionalidad, podrían aparecer dependencias con ciertas entidades del programa, caso en el que sería conveniente definir también una clase base *EntidObjetBase*.

Aspecto ContFEntrnOHardInteractivo

El *aspecto ContFEntrnOHardInteractivo* está ideado para aquellos casos en que se requiere una mayor interacción con el sensor, ampliando por tanto el abanico de *puntos de unión* a interceptar por el *punto de corte ActivarConsultaSensor*. Sin ir más lejos, en los dos casos de estudio tratados se establecen como *puntos de unión* los métodos encargados de gestionar los eventos de interacción con el dispositivo móvil. Cabe señalar que, a diferencia del *aspecto ContFEntrnOHardBajoDemanda*, el *consejo* asociado tan sólo se encarga de disparar la comunicación con el sensor. Tanto es así que lo más habitual es que el método *discernirValorSensor* se presente vacío.

En consecuencia, tanto el análisis para discernir si es conveniente reflejar una adaptación o no, como la ejecución de la adaptación propiamente dicha (invocación al método *aplicarAdaptacion* de la clase *ReguladorPantalla*) se articulan a través de un nuevo *punto de corte* específico, denominado *ControlarCambios*. Éste se activa cada vez que el valor correspondiente al factor objeto de tratamiento (valor gestionado por la clase *GestorFactor*) sufre una modificación suficientemente significativa (designador *set* en *AspectJ*, tal y como aparece en el diseño –figuras 8.20 y 8.21-).

Este *aspecto* es el candidato apropiado para el tratamiento de aquellos factores ambientales que requieren un control de cambios frecuente. Un ejemplo es el trabajado en los casos de estudio –la luz exterior-, donde a través de la interacción del usuario se detecta que se está haciendo uso del sistema, y con el propósito de hacer la lectura agradable se regula el brillo de la pantalla de manera automática.

Aspecto ContFEntrnOHardBackground

Por último, el aspecto *ContFEntrnOHardBackground* está pensado para los casos en que se cree conveniente que el control del nivel del factor a tratar se realice de manera constante y cíclica, complementando el que se lleva a cabo a través de los *puntos de corte* mencionados anteriormente. La idea es combinar la consulta realizada por el propio *aspecto* con una consulta en background llevada a cabo por un *thread*, a activar cada cierto tiempo. Es el propio *aspecto* quien lanza el *thread* a ejecución en el paso de inicialización (constructora del *aspecto*) el cual, de forma paralela, va activando la comunicación con el sensor.

Igualmente, tanto el análisis para discernir si es conveniente reflejar una adaptación o no, como la ejecución de la adaptación propiamente dicha se articulan a través del *punto de corte ControlarCambios*.

Ejemplos donde este aspecto sería adecuado son, de nuevo, el trabajado con la luz exterior en el caso de estudio *'Lector de Noticias'*, donde además de comunicarse con el sensor al interactuar con el dispositivo, también se aplica un control cada cierto tiempo. Otro ejemplo sería el de regular el volumen de una reproducción al sonido de fondo, donde un control constante puede ayudar a mantener la atención a lo largo de la transmisión.

En el *Apéndice C* se presenta el diagrama de clases completo (*capas aspectual* y *sensible al contexto*) para esta componente, donde también queda reflejado cómo afecta a la aplicación base. Pueden consultarse dos diagramas para las dos versiones desarrolladas: J2SE y J2ME.

8.2.4.3. Consideraciones relativas a la implementación

La implementación de esta componente del framework para un caso concreto consiste en completar los siguientes elementos de programa:

- uno de los *sub-aspectos*, aquel que mejor se ajuste a las necesidades a tratar. En concreto, consiste en concretar el método *discernirValorSensor* y, dependiendo de los casos, los *puntos de corte ActivarConsultaSensor* y el consejo del *punto de corte ControlarCambios*. El resto de código viene ya proporcionado por el propio *aspecto*, puesto que el código de los *consejos* tan sólo hace uso de las clases de la *capa sensible al contexto* que, como ya se ha mencionado, a su vez es genérico.
- el método *aplicarAdaptacion* implementado por la clase *ReguladorPantalla*, el cual concentra la operativa propia de la adaptación.

- la clase *Sensor*, encargada de la comunicación con el sensor o, en su caso, de su simulación. Como ya se ha visto, esta clase implementa la interfaz *perceptorEntrnOHardw*, con lo que deberá ir provista de la definición del método *obtenerValorSensor*.

8.2.5. Consideraciones al respecto de la aplicabilidad del FPI

A continuación se recogen una serie de consideraciones a tener en cuenta respecto a al aplicabilidad del FPI como son la documentación que acompaña al framework, las consideraciones en cuanto a la interferencia o conflictos entre *aspectos* y, por último, un pequeño análisis de la satisfacción de requisitos como infraestructura del contexto.

8.2.5.1. Documentación

Sin duda, parte de la clave del éxito de aplicación de un framework está supeditada a la documentación que éste proporcione, en especial en la parte en que se describen los pasos para su correcta instanciación.

En el caso del FPI es interesante, además, identificar y dar a conocer qué partes del sistema base deben explorarse y conocerse en detalle, a efectos de determinar (1) qué partes deben interceptarse para establecer los *puntos de corte*; (2) qué clases deben ser consideradas como *entidades objetivo*; (3) de qué métodos disponen las *entidades objetivo*, a fin de deducir si es necesario ampliar su estructura o el número de operaciones, con objeto de poder tanto capturar la información requerida como aplicar la adaptación para la que está destinada la componente en cuestión; (4) en qué casos se requiere incorporar clases genéricas para las *entidades* o pantallas *objetivo*; y por último (5) conocer en profundidad cómo se lleva cabo la funcionalidad que se desea adaptar, a fin de incorporar exactamente la parte que se desea aumentar de la aplicación, a codificar en los correspondientes *consejos* de los *aspectos*. En efecto, estos datos resultan imprescindibles en la instanciación del framework.

Puesto que la aplicación del framework no escapa de la necesidad de tener cierto conocimiento al respecto de las técnicas de POA, resulta útil proporcionar también ciertas nociones y explicaciones al respecto –en concreto de su aplicabilidad en el lenguaje *AspectJ*-, a fin de familiarizar al usuario del framework con los conceptos manejados.

8.2.5.2. Interferencia entre aspectos

Otro aspecto a tener en cuenta es el de *la interferencia entre aspectos*. Cabe esperar que una misma aplicación incorpore no una, sino varias componentes de adaptación a

distintas *restricciones de tiempo real*. En esos casos, la coexistencia de diversos *aspectos* en la *capa aspectual* introduce la posibilidad de provocar conflictos entre *aspectos*. Este es un problema presentado en el *Capítulo 6* (véase *Capítulo 6*; *sección 6.2.3.3.4.*). Puesto que el enfoque de adaptación aplicado en todo momento está preestablecido de antemano, una estrategia de tiempo de compilación es suficiente para resolver esta cuestión. En efecto, la inclusión del código correspondiente a la nueva funcionalidad se produce siempre de manera estática, de acuerdo a unas necesidades y tratamientos predefinidos. Como ya ha sido explicitado a lo largo del documento, bajo el enfoque de la *visión dicotómica de plasticidad* las situaciones no previstas en la operativa del MPI son delegadas en el *servidor de plasticidad*, el cual, si es necesario, introduce nuevos módulos de *aspectos* en el ejecutable (véase *sección 8.4.2.*). Bajo estas premisas, la *precedencia entre aspectos* puede ser también preestablecida de antemano.

De cualquier modo, ésta es una cuestión importante que merece la atención oportuna. Tal y como se comenta en el *Capítulo 6*, *AspectJ* proporciona un mecanismo de *precedencia entre aspectos* explícito muy sencillo que permite establecer el orden de intervención de los distintos *aspectos* implicados en tiempo de compilación. La construcción sintáctica correspondiente se detalla en el *Apéndice A*. La estrategia más adecuada para incorporar la *precedencia entre aspectos* en una determinada aplicación es, de nuevo, a través de la incorporación de un *aspecto* específico para ello, a ubicar también en la *capa aspectual*.

8.2.5.3. Análisis de la satisfacción de requisitos

En cuanto al cumplimiento de requisitos identificados por Dey [Dey00] para las infraestructuras del contexto (véase *Capítulo5*; *sección 5.2.2.*), a continuación se relacionan aquellos que son satisfechos por el FPI.

- *separación de conceptos*, puesto que, en efecto, la información contextual utilizada por las aplicaciones no proviene directamente de los sensores, sino que atraviesa uno o dos niveles de abstracción, de manera que la adquisición del contexto se realiza de forma separada al uso que se hace del mismo por parte de la *capa aspectual*. De hecho, el proceso de adquisición del contexto le resulta transparente, no sólo a la *capa lógica* del sistema, sino también a la *capa aspectual*, mediante el uso de una interfaz común.
- *especificación del contexto*, puesto que proporciona un mecanismo para indicar al *servidor de plasticidad* en qué información contextual está interesada la aplicación. A través del envío de peticiones de adaptación, la plataforma cliente da a conocer la situación en la que se encuentra durante la ejecución, constituyendo un soporte

adecuado para especificar en qué contexto está interesada una aplicación en cada momento. El *servidor de plasticidad*, a su vez, incorpora los componentes del contexto más apropiados para las necesidades contextuales explicitadas.

- *interpretación del contexto*, es decir, un soporte para transformar uno más tipos de contexto en otro distinto, resultante de la combinación de ellos. Aunque a priori tan sólo está prevista la *precedencia entre aspectos*, tal y como ya ha sido mencionado, se concibe el framework como una API abierta y factible de ir ampliando y evolucionando sus posibilidades. En consecuencia, la combinación explícita de distintos tipos de contexto podría incorporarse como un módulo más en la librería de *aspectos* y clases convenientemente jerarquizada. No obstante, a nivel estático sí se tiene en cuenta la confluencia de todos los factores del contexto, a través del proceso llevado a cabo en el *servidor de plasticidad*.

- *almacenamiento del contexto*, puesto que en todos los componentes está prevista esta utilidad, a fin de que la información acerca del historial pueda ser utilizada para establecer tendencias y predecir futuras necesidades del contexto, responsabilidades que recaen sobre el *servidor de plasticidad*.

Los requisitos relacionados con el desarrollo de middlewares del contexto (descubrimiento de recursos y transparencia en el mecanismo de comunicación), así como el de *disponibilidad constante para la adquisición del contexto* no atañen a la infraestructura de la plataforma cliente, concebida para resolver el contexto de forma autónoma e individual por parte de cada plataforma cliente.

8.3. Visión de conjunto de la infraestructura y del proceso de plasticidad propuestos

A lo largo de este documento se han descrito en detalle los dos motores en los que se divide la infraestructura de plasticidad propuesta bajo el enfoque de la *visión dicotómica*, a fin de resolver los cambios contextuales de forma efectiva explotando las dos facetas de plasticidad identificadas (la *explícita* y la *implícita*), mutuamente complementarias entre sí. Dichos motores son los denominados MPE y MPI respectivamente, responsables de proporcionar plasticidad durante el uso del sistema actuando en dos niveles de operación diferenciados. La repartición de responsabilidades entre ambos motores no responde a un patrón rígido y común a todos los casos, sino que se promueve una total flexibilidad.

En esta tesis se propone una arquitectura software para la construcción de MPIs (ésta se presenta en el *Capítulo 6*). En cuanto a los MPEs, que se asimilan a las herramientas

basadas en modelos, no se propone una nueva herramienta, sino que se presenta un marco conceptual para la construcción de MPEs denominado *Framework de Plasticidad Explícita Colaborativo* –FPE Colaborativo. Como marco conceptual, no constituye una herramienta operativa, y por lo tanto no entra en juego en el proceso operativo de plasticidad. Debe entenderse como un instrumento de estudio y desarrollo de este tipo de herramientas, tal y como se describe en el *Capítulo 4*.

Adicionalmente, se ha desarrollado el FPI, presentado en este capítulo. El FPI interviene en la fase de diseño, una vez se ha construido la IU de partida del sistema. Como framework personalizable en componentes del contexto, sirve de soporte en la incorporación de capacidades adaptativas en el sistema, un paso que se lleva a cabo previo a la entrega del sistema a la plataforma cliente, listo para su uso. Su intervención es crucial en la preparación del sistema, tanto en la primera versión como una vez que ha iniciado la interacción y uso del sistema.

Es precisamente el papel que desempeña el FPI en el proceso llevado a cabo en el *servidor de plasticidad*, así como las etapas en que se divide dicho proceso, el objetivo de esta sección. Se pretende con ello explicitar el proceso global de plasticidad y proporcionar una visión integradora de la operativa conjunta.

En una primera sub-sección se describe el proceso de plasticidad en su conjunto, esto es, la operativa llevada a cabo entre la plataforma cliente y el servidor, a fin de satisfacer todas las necesidades contextuales durante el uso del sistema. La siguiente sub-sección centra el foco de atención en el *servidor de plasticidad*, con el propósito de detallar los pasos y componentes que intervienen, a fin de obtener no sólo una *IU específica* adaptada a las condiciones solicitadas por la plataforma cliente, sino un sistema completamente acondicionado a las nuevas circunstancias, listo para ser entregado y desplegado en la plataforma cliente.

8.3.1. Proceso de plasticidad en su conjunto

La *visión dicotómica*, que se enmarca en un modelo de arquitectura cliente-servidor, fomenta la complementación entre los dos niveles de operación de plasticidad de la forma más natural y flexible posible, esto es, sin determinar de forma rígida qué tipo de adaptaciones corresponden a cada motor. La repartición de responsabilidades de adaptación está supeditada a la autonomía y capacidad de cada plataforma en el lado del cliente para asumir los distintos tratamientos. El objetivo que hay detrás de este enfoque es el de buscar la máxima autonomía y efectividad, a fin de responder a los cambios contextuales en tiempo real siempre que sea posible, evitando una fuerte dependencia de la red. Se pretende alcanzar la flexibilidad suficiente como para que la plataforma cliente tan sólo tenga que

recurrir al MPE cuando no puede asumir de forma autónoma la adaptación a un cambio contextual, por la envergadura que comporta. Así, cada tipo de dispositivo puede ajustar su nivel de autonomía en función de sus capacidades, potenciando un uso razonable de la red, y tendiendo hacia un equilibrio entre el nivel de adaptación y el uso que se hace de la misma.

Este enfoque difiere de un enfoque monolítico donde existe un único mecanismo de adaptación que trata todas las situaciones por igual, o bien de otros enfoques donde cada motor opera de manera independiente.

Con el propósito de que la operación de ambos motores se complemente en todo momento, a fin de sacar el máximo partido a los beneficios de combinar ambos niveles de operación, el MPE y el MPI actúan iterativa y alternativamente, retroalimentándose mutuamente, sucediéndose etapas de producción –en algunos casos reconfiguración de una IU ya existente- y uso –manejo en tiempo de ejecución- de la IU, en función de las necesidades que puedan ir surgiendo a lo largo de la interacción con el sistema. Se trata de un proceso cíclico que no está establecido de antemano, sino que depende de la evolución que siga la interacción, y de las circunstancias contextuales que se vayan encontrando, entre las cuales se incluyen las condiciones de grupo en el caso de actividades colaborativas. Para ello es esencial garantizar una retroalimentación adecuada entre los modelos del contexto de ambos lados de la arquitectura. De hecho, esta retroalimentación constituye la clave para conseguir que el proceso de plasticidad tome en consideración en todo momento los cambios contextuales experimentados durante la ejecución, con el fin de obtener la respuesta más apropiada a la situación propia de cada momento. El objetivo es evitar en la medida de lo posible *discontinuidades de plasticidad*, garantizando un nivel apropiado de plasticidad. Todo este proceso queda detallado en el *Capítulo 2* (véase *Capítulo 2*; *sección 2.1.5.*).

En lo sucesivo se denomina *ciclo de plasticidad* al periodo de tiempo comprendido entre dos peticiones de adaptación sucesivas procedentes de la plataforma cliente, abarcando por tanto la generación de una IU, la entrega del sistema a la plataforma cliente y su uso hasta que surge una nueva necesidad contextual no resoluble en la propia plataforma o dispositivo de interacción.

8.3.2. Proceso llevado a cabo en el servidor de plasticidad

A continuación se describen las etapas que componen el proceso de construcción de un MPI específico, a resolver en cada *ciclo de plasticidad* por parte del *servidor de plasticidad*.

1. *Construcción de la IU específica por parte del MPE* para unas necesidades contextuales determinadas, las cuales han sido transmitidas por la plataforma cliente a través de una petición, y que han dado lugar a la actualización de los *modelos contextuales* en el MPE, como paso previo a la producción de una nueva IU, tal y como se detalla en el *Capítulo 4* (véase *Capítulo 4*; *sección 4.1.5.1.*).

   Una vez obtenida la nueva IU, no es suficiente con enlazar ésta con el resto de la aplicación a fin de ser entregada. El código resultante consistiría en una aplicación común con la única particularidad de que la IU ha sido adaptada estáticamente a las condiciones contextuales expresamente especificadas. El sistema no sería capaz de proporcionar adaptaciones proactivas, ni por supuesto *mecanismos locales de consciencia de grupo*, en el caso de tratarse de una actividad colaborativa. No se comportaría, por tanto, como un sistema sensible al contexto. Esto conllevaría a una fuerte dependencia del servidor, al tener que recurrir a él ante cualquier *variación dinámica del contexto de uso* (véase *Definición 2.3*; *Capítulo 2 – sección 2.1.3.*). Por lo tanto, a fin de explotar también la faceta proactiva de la plasticidad deben darse los pasos oportunos para embeber un MPI en el código de la aplicación que se ajuste al sistema y necesidades del momento. Es por ello que, previo a la entrega a la plataforma cliente es necesario llevar a cabo las dos etapas siguientes (en ocasiones es suficiente con las etapas 3 y 4).

2. *(no imprescindible) Personalización*, esto es, instanciación y acoplamiento del FPI –disponible en el *servidor de plasticidad*- a la aplicación y a las necesidades contextuales requeridas. Se está haciendo referencia a la construcción y acoplamiento de las oportunas componentes del contexto a partir del framework genérico. Este paso se denomina *derivación del MPI* que, por supuesto, se lleva a cabo también en el *servidor de plasticidad*, como parte de la adaptación estática (etapa de diseño).

   Esta etapa puede obviarse siempre y cuando el MPI no requiera ser actualizado, es decir, siempre que las *restricciones de tiempo real* no hayan sufrido cambios importantes durante el *ciclo de plasticidad*. En estos casos puede ser reutilizado el mismo MPI del que se ha hecho uso en el ciclo anterior.

   Una vez disponibles todos los componentes: *capa lógica* por un lado, y *capas aspectual* y *sensible al contexto* por otro (reutilizados del ciclo anterior o instanciados en este mismo *ciclo de plasticidad*), se pasa a la etapa siguiente.

3. *Entrelazado*, correspondiente al paso de compilación y entretejido de *aspectos* y clases por parte del *weaver* (entretejedor de *aspectos*). Este paso puede ser automatizado mediante el uso de un archivo de *script* o de procesamiento por lotes. El resultado es, ni más ni menos, el MPI solicitado desde la plataforma cliente, esto es, la aplicación

adaptada a las condiciones contextuales y provista de capacidades adaptativas, lista para su uso. Tan sólo queda llevar a cabo un último paso.

4. *Empaquetado y entrega* a la plataforma destino, a fin de proseguir con el uso de la aplicación hasta completar un nuevo ciclo.

Es importante remarcar que el paso 2, que es precisamente el que requiere mayor atención por parte del diseñador, no es un paso imprescindible. Esto queda reflejado en la figura 8.22 mediante el uso de corchetes en el etiquetado de las etapas (la etapa 2 en este caso). Las situaciones en que es conveniente construir un nuevo MPI son cuando o bien se producen cambios significativos en el entorno que dan lugar a que intervengan nuevas componentes para el tratamiento de nuevos atributos, o bien cuando se produce un cambio en la plataforma cliente. En este último caso, las necesidades previsibles de adaptación al contexto varían significativamente. Por ejemplo, si se pasa de una plataforma fija a un dispositivo móvil, los factores del entorno a considerar se amplían, e incluso puede ser necesario incorporar alguna componente para el tratamiento de las restricciones en recursos hardware, dadas las nuevas limitaciones. También es adecuado recurrir a este paso cuando se desea cambiar la parte adaptativa de la aplicación, por ejemplo, haciendo entrar en juego nuevas componentes de adaptación al usuario. En ocasiones esta fase puede ser destinada a ajustar el valor de ciertos parámetros relacionados con ciertos umbrales o valores de partida (e. g. el tamaño máximo de un archivo, la extensión en días del periodo de muestreo, umbral de variación de un atributo del contexto, etc.), sin necesidad de llevar a cabo todo el proceso de construcción del MPI.

La etapa 3 de entrelazado es insalvable si se quiere dotar a la aplicación de sensibilidad al contexto. De no haber desarrollado las *capas adaptativa* y *sensible al contexto* (etapa de *derivación del MPI*) en este ciclo, el código que se entrelaza con la aplicación base (*capa lógica*) es el que ya formaban parte de la fase de ejecución anterior (parte inferior izquierda de la figura 8.22). Ello significa que los MPIs generados en el *servidor de plasticidad* se mantienen de un *ciclo de plasticidad* a otro. Esto sólo es factible gracias al alto grado de ortogonalidad y reutilización proporcionado por los *aspectos*, que permite separar el código de cada capa sin problema alguno, al no existir acoplamiento entre ellas. Esta doble alternativa queda reflejada en la figura 8.22 delimitando el número correspondiente a esta fase (la fase 3) entre paréntesis. Como se observa en la figura 8.22, la entrada al *weaver* puede proceder de la fase de *derivación del MPI*, o bien sin pasar por ella, esto es, reutilizando las capas del ciclo previo (parte inferior izquierda).

Aunque la etapa de *derivación del MPI* requiere ser desarrollado manualmente, puede ser automatizada la parte de seleccionar los componentes (*aspectos* y clases) del framework

a ser utilizados en el proceso de especialización, tomando en consideración la información contextual recibida de la plataforma cliente, y que ha sido reflejada en los modelos contextuales (modelo de usuario, espacial, de entorno, de plataforma y de grupo). Ese es el motivo por el que dichos modelos están conectados con la realización de esta fase en la figura 8.22. En los casos en que esta etapa se obvía el proceso en el servidor queda casi completamente automatizado, por lo que en determinados casos podrá ser resuelto incluso sin interrumpir la ejecución en la plataforma cliente, es decir, sin discontinuidad técnica.

Resulta interesante resaltar aquí que este proceso favorece un trabajo en equipo, permitiendo la división de tareas en un equipo de software. Así, se pueden distinguir hasta tres roles distintos: (a) el que se encarga de supervisar y hacer los refinamientos oportunos a lo largo del proceso de generación de la IU en el MPE, al que se le denomina *diseñador de plasticidad explícita*; (b) el que se encarga de diseñar el FPI, al que se le denomina *diseñador de plasticidad implícita*; y por último (c) el que se responsabiliza de personalizar el FPI a sistemas concretos. Este es el miembro del equipo sobre el que recaen más responsabilidades, puesto que no sólo debe tener un dominio de la aplicación objetivo sino que, además, debe estar familiarizado con el FPI, a fin de realizar la instanciación y acoplamiento con la aplicación. A este tercer rol se le ha denominado *diseñador de MPIs*. Todos estos roles aparecen también reflejados en la figura 8.22. Adicionalmente está el rol del *desarrollador de aplicaciones*, cuya intervención tan sólo es necesaria si el código núcleo de la aplicación requiere un cambio sustancial para ser migrado a una plataforma distinta.

Otra observación es que la jerarquía de *aspectos* y objetos representadas en la figura 8.22 (parte superior derecha) simboliza una librería jerárquica de *aspectos* y clases relacionadas entre sí, esto es una API organizada por categorías contextuales. Efectivamente, se ofrecen componentes semi-completas para distintos atributos del contexto (entorno y restricciones hardware, distintos objetivos de personalización al usuario y componentes de apoyo al trabajo en grupo) a las que sucesivamente se le pueden ir incorporando nuevas componentes, conforme nuevos tratamientos para necesidades contextuales o adaptativas vayan siendo desarrollados. En definitiva, se trata de una API abierta que permite ir incorporando progresivamente nuevos módulos. En definitiva, lo que constituye verdaderamente un framework es el conjunto de clases y *aspectos* con los que derivar el MPI objetivo, una vez han sido seleccionados de la librería jerárquica.

Una última observación a realizar es la siguiente. Como se ha podido observar, es ya desde la fase de diseño que se está contribuyendo a la obtención de adaptaciones proactivas, puesto que en el proceso de construcción del MPI específico para cada situación se están anticipando las necesidades contextuales previsibles en tiempo de ejecución, previo a su

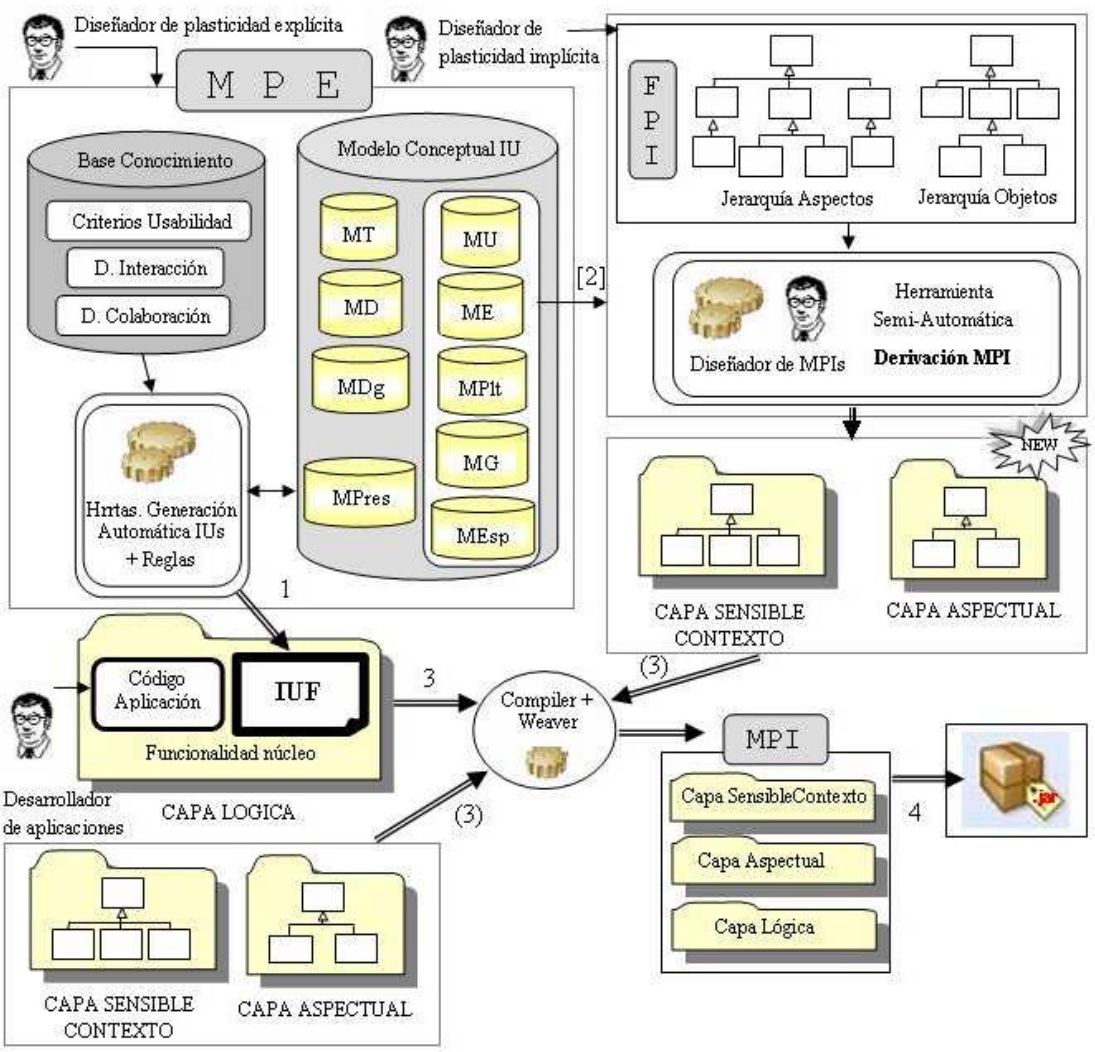


Figura 8.22: Proceso llevado a cabo en el *servidor de plasticidad* para la producción de un MPI.

entrega al dispositivo cliente. Ese es el modo como bajo el enfoque propuesto en esta tesis se contribuye a una anticipación a los cambios contextuales en la propia etapa de diseño. En definitiva, el hecho de decidir qué *aspectos* del contexto interesa integrar en cada caso, acción que se lleva a cabo en el *servidor de plasticidad*, demuestra que ya desde la fase de diseño se está contribuyendo a la posterior ejecución de las adaptaciones proactivas, según las necesidades previsibles. De hecho, ésta constituye una de las hipótesis planteadas en el *Capítulo 1* (véase *Hipótesis* 5; Capítulo1 - *sección 1.3.*).

8.4. Resumen y conclusiones del capítulo

En este capítulo se ha descrito el framework genérico para dotar a las aplicaciones de capacidades de adaptación al contexto en tiempo de ejecución, capacidades que, siguiendo el enfoque de la *visión dicotómica*, se integran en la propia aplicación, a fin de que éstas se muestren autónomas en la ejecución de las mismas, proporcionando en definitiva lo que ha sido definido en esta tesis como *plasticidad implícita*. De ahí que éste sea nombrado como *Framework de Plasticidad Implícita*.

Se trata de un *framework de caja blanca*[8], puesto que soporta extensibilidad proporcionando clases y *aspectos* base a ser extendidos, provistos de métodos y *puntos de corte* predefinidos que hacen posible la personalización del framework en aplicaciones concretas. En general, el uso de este tipo de frameworks requiere que los desarrolladores dispongan de un conocimiento minucioso de su estructura interna pero, en contrapartida, proporcionan un soporte adecuado para adaptar la funcionalidad interna del mismo permitiendo satisfacer necesidades específicas de las aplicaciones, así como nuevos requisitos que no han sido previstos a priori en la fase de diseño. Este factor otorga a este tipo de frameworks de una gran flexibilidad en relación con los *frameworks de caja negra*, donde no pueden definirse otros comportamientos que los estrictamente suministrados [FS97].

Otra de las propiedades que caracterizan los *frameworks de caja blanca* es el hecho de que se utiliza más herencia que composición. Precisamente es la herencia el mecanismo que proporciona flexibilidad cuando el rango completo de funcionalidad no puede ser anticipado, como es el caso en las aplicaciones sensibles al contexto.

Bajo la opinión personal del autor de esta tesis, el enfoque propuesto para dotar a las aplicaciones de sensibilidad al contexto, descrito en este capítulo, resulta novedoso, altamente aplicable y en el que se han seguido los principios de un buen diseño de software para su desarrollo, haciendo prevalecer aspectos como la ortogonalidad, la modularidad y la genericidad de código, a fin de alcanzar un nivel de abstracción y reutilización aceptable. En particular, el nivel de ortogonalidad es especialmente destacado en el caso de componentes de adaptación al entorno y recursos hardware, al tratar con valores que proceden del mundo exterior. Este factor facilita tanto su aplicabilidad como su reutilización a distintos factores.

Se considera novedoso porque tal y como se deduce de la revisión del estado del arte llevada a cabo en los *Capítulos 5* y 6, existen pocos frameworks de aplicación para la obtención de *aplicaciones sensibles al contexto* (la mayoría de trabajos consisten en el

[8]La distinción entre frameworks de caja blanca y frameworks de caja negra responde al criterio de personalización de los mismos, y cuya categorización es debida a [JF88].

desarrollo de arquitecturas o middlewares), y los existentes no se corresponden con las condiciones de aplicabilidad para las que ha sido concebido el *Framework de Plasticidad Implícita* (véase *Capítulo 6*; *sección 6.2.*). Otros intentos por lograr reutilizar *aspectos* se presentan de manera informal como casos prácticos que, por otro lado, señalan como situaciones de reutilización problemática algunos puntos que, por primera vez, quedan resueltos utilizando el enfoque seguido en el *Framework de Plasticidad Implícita*, y que ha sido explicado en detalle en este capítulo.

Se considera altamente aplicable porque, en primer lugar, permite que cualquier aplicación pueda embeber componentes del contexto, independientemente del dominio de aplicación o la plataforma a utilizar, pudiendo aquélla fácilmente ser reconfigurada para unas necesidades contextuales distintas. Por otro lado, a pesar de ser un *framework de caja blanca*, el grado de reutilización de código es también elevado, lo que facilita también su instanciación, favoreciendo que la curva de aprendizaje requerida para la comprensión y uso del framework resulte compensatoria. En efecto, la proporción de código a especializar corresponde a porciones muy concisas que, en la mayoría de los casos, consisten en código puramente *orientado a objetos* (en particular, métodos a ser especializados). Cabe recordar que buena parte de la implementación del código de tratamiento del contexto queda delegada en la *capa sensible al contexto*. Todo ello facilita su aplicación, a pesar de que el usuario tenga tan sólo leves nociones acerca de las técnicas de POA. De hecho, ciertas nociones ineludibles en el uso de POA pueden ser proporcionadas en la propia documentación del framework.

A pesar de que parte de la instanciación consiste en saber identificar qué puntos de la ejecución del programa deben ser interceptados, así como cuál es la información que debe ser capturada, eso no significa tener que dominar la manera como se utiliza dicha información, a fin de conseguir el tratamiento deseado. En conclusión, aprender a usar el FPI, no significa comprender cómo ha sido construido, como suele ocurrir en los frameworks de caja blanca.

El salto hacia un mayor grado de reutilización que proporciona la combinación de *anotaciones de metadatos* con el uso de *aspectos* es decisivo de cara a la reutilización y facilidad en su aplicación, puesto que evita la necesidad de extender *aspectos* . De hecho, es conocido que el uso de la facilidad de *metadatos* aporta el potencial de simplificar y mejorar muchas áreas de desarrollo de aplicaciones, entre las que se enumeran la implementación de frameworks [Lad05]. Cabe esperar que esta facilidad esté pronto disponible también en la modalidad J2ME.

Los frameworks proporcionan código fuente listo para ser utilizado, al tiempo que identifican y localizan aquellas partes que requieren especialización. Una vez superada la curva

de aprendizaje requerida para su comprensión, es obvio que el tiempo empleado para llegar al producto final se reduce considerablemente, sobretodo si el nivel de reutilización es elevado. Ha sido reconocido que el uso de frameworks constituye un mecanismo para simplificar el diseño y desarrollo de aplicaciones en general [FS97], y groupware en particular [SKSH96].

No obstante, el desarrollo de frameworks constituye un área relativamente joven y, en consecuencia, aún quedan muchos problemas por resolver, tales como la documentación, el análisis del coste que supone su desarrollo, de cara a estimar la compensación del esfuerzo empleado en su construcción, así como la valoración de los beneficios que supone su uso [FSJ99].

Gracias al alto grado de modularidad con el que ha sido construido el FPI, su despliegue puede ser organizado como una librería jerarquizada de *aspectos* y clases, cuya amplitud y campo de aplicación está supeditada a la cantidad y variedad de componentes y tipos de contexto ofrecidos. Se concibe, por tanto, como un *toolkit* abierto susceptible de seguir evolucionando, al tiempo que amplía sus mecanismos de adaptación y tratamientos predefinidos del contexto. Con el propósito de hacer factible este despliegue, es esencial no sólo aplicar unas estrategias adecuadas de genericidad y reutilización, sino también ser estratégico en el modo como estructurar la jerarquía de *aspectos* y clases a construir, de acuerdo a la gran variedad de necesidades contextuales y de mecanismos de adaptación que pueden ser requeridos.

Tal y como se muestra en la *sección 8.4* anterior, el FPI se integra en un proceso de plasticidad basado en el enfoque de la *visión dicotómica*, el cual pretende ofrecer un cierto nivel de sistematización, y donde el FPI tiene la misión de facilitar el diseño de componentes contextuales y de apoyo al trabajo en grupo adecuadas a sistemas particulares.

Conscientes de la importancia de una buena documentación para los frameworks, resulta conveniente destinar parte del esfuerzo en su elaboración, acompañando la explicación de cada componente con una exposición clara y concisa de los pasos requeridos para su instanciación.

Capítulo 9

Conclusiones

"Me interesa el futuro porque es el sitio donde voy a pasar el resto de mi vida."

Woody Allen

9.1. Conclusiones Generales

La creciente proliferación de dispositivos computaciones, así como la posibilidad de interaccionar en escenarios cada vez más diversos, caracterizados por cierta movilidad, ha creado la necesidad de que las aplicaciones puedan ser ejecutadas en múltiples plataformas y por distintos perfiles de usuarios, a la vez que se espera una cierta *sensibilidad* por parte del sistema a las condiciones contextuales que caracterizan los distintos entornos físicos y de trabajo.

Es bien conocido que la computación móvil impone una serie de retos importantes en el diseño y desarrollo de IUs. Cada plataforma móvil tiene sus propias restricciones y capacidades. La falta de uniformidad no sólo en hardware, sino también en software, ocasionado por el hecho de que cada fabricante se decide a implementar sus versiones propias de perfiles y JVMs, ajustadas específicamente a sus modelos, obliga a los desarrolladores a optar entre dos posturas extremas: restringir el grupo específico de terminales móviles para los que desarrollar su aplicación, o bien considerar únicamente aquellas características que cumplen todos los terminales, lo que da lugar a aplicaciones pobres. Este problema es bien conocido como *fragmentación de APIs* [AMC+05].

Aunque los nuevos estándares para móviles Java Mobile Service Architecture (MSA) y MIDP 3.0, tienen como objetivo definir un conjunto de funcionalidades estándar, diferenciando una base común de las tecnologías más avanzadas, sorprendentemente están siguiendo el mismo modelo[1] caracterizado por una explosión de APIs. Las posibilidades hardware y de servicios de los terminales van en aumento, lo que hace difícil creer que esta carrera por conquistar el mercado pueda llegarse a frenar dando fin a esta fragmentación del mercado.

Otra problemática relacionada es la que caracteriza la Web móvil, que aunque resulta muy atractiva, plantea numerosos problemas por resolver. En este caso hay que sumar a la heterogeneidad de dispositivos, cuya variedad en tamaños de pantalla y posibilidades gráficas es desbordante, la dificultad de tratar con varios lenguajes de marcado, el ancho de banda y la entrega de contenidos especialmente ajustada a las necesidades propias de un contexto móvil. Por último, hay que añadir el handicap impuesto por el coste que supone hoy en día la Internet móvil.

En la búsqueda de estrategias que proporcionen flexibilidad es esencial disponer de mecanismos para preservar la usabilidad. De poco sirve una IU adaptable si no es usable. Este es el aspecto diferenciador de la *plasticidad*. Aunque se trata de un campo incipiente de menos de una década de antigüedad, la problemática asociada irá haciéndose cada vez más notoria en los próximos años y, en consecuencia, este campo irá ganando relevancia.

La solución aportada en esta tesis está ideada y desarrollada en base a la separación conceptual del concepto de plasticidad, la cual ha sido identificada y definida como fruto del estudio llevado a cabo, y que ha dado lugar a enunciar dos sub-conceptos distintos de plasticidad. En consecuencia, este documento ha sido estructurado también en base a dicha distinción. Como resultados se presentan dos artefactos a ser utilizados respectivamente en los dos niveles de operación identificados, cada uno específico para su ámbito, con los que se ha tratado de cubrir algunas de las limitaciones detectadas en la literatura propia de cada área.

Así, el campo de la *plasticidad explícita* constituye un campo ciertamente maduro, como demuestra la disponibilidad de algunas herramientas basadas en modelos comerciales, aunque sigue presentando aspectos que requieren cierta atención, como son el tema de la preservación de la usabilidad, la posibilidad de anticipación a los cambios contextuales ya desde una fase de diseño y la propagación de los cambios ocurridos durante la ejecución. Se ha propuesto un marco conceptual como instrumento de referencia para el estudio de herramientas basadas en modelos, y que, además, proporciona una orientación en el desarrollo de IUs plásticas. Se ha hecho un esfuerzo por comprender diversas herramientas

[1] http://www.mas34.net/2007/04/24/java-msa-o-midp-30- %C2 %BFson-la-solucion/

y aproximaciones existentes, a fin de proponer un esquema uniforme para representar los distintos enfoques. Hasta ahora existía un único marco de referencia, el cual deja muchos aspectos sin considerar.

Como aspecto novedoso se han integrado las consideraciones relativas al grupo de trabajo, en un intento por incorporar las peculiaridades y necesidades intrínsecas de los entornos colaborativos en el proceso de construcción de IUs.

Por otro lado, el área de desarrollo de *aplicaciones sensibles al contexto* (campo que se equipara con el de la *plasticidad implícita*) es un área relevante y bien conocida cuyo desarrollo se caracteriza por la resolución de una misma problemática de manera recurrente, motivado por las variaciones importantes que caracterizan el ámbito. Ante la falta de una solución genérica y reutilizable, se ha desarrollado un framework que resuelve las expectativas dinámicas de adaptación bajo el enfoque de la *visión dicotómica*. Este framework ha sido denominado *Framework de Plasticidad Implícita* (abreviadamente FPI). Es altamente reutilizable al no estar restringido a un dominio o a una plataforma, y además está estructurado de tal forma que puede seguir evolucionando para la incorporación de nuevas necesidades contextuales y mecanismos de adaptación. Puede ser instanciado codificando tan sólo una pequeña parte del código que, en su mayoría, corresponde a código *orientado a objetos*. El FPI ha estado construido siguiendo la arquitectura software propuesta en esta tesis, la cual minimiza el acoplamiento de los componentes del contexto con el núcleo de la aplicación. Se basa en el uso de técnicas de Programación Orientada a Aspectos (POA).

A nivel global se propone una infraestructura integradora, enmarcada en una arquitectura cliente-servidor que, en lugar de estar centrada en el servidor, como suele observarse en la mayoría de enfoques encontrados en la literatura, intenta ofrecer una estrategia equilibrada tratando de proporcionar una distribución equitativa de las responsabilidades de adaptación a ambos lados de la arquitectura. La meta es la de reunir las ventajas tanto de un enfoque centralizado como de un enfoque totalmente distribuido. Cabe resaltar que éste fue el planteamiento inicial enunciado a través de la *Hipótesis 2*. Las pruebas realizadas demuestran la autonomía y la proactividad esperadas en dispositivos compactos. Por otro lado, las características intrínsecas a una arquitectura cliente-servidor proporcionan un mecanismo para la propagación de cambios producidos durante la ejecución al *servidor de plasticidad*, así como para compartir información de *awareness*.

Con el propósito de proporcionar una perspectiva global de la infraestructura propuesta se ha mostrado también una visión integradora de los distintos motores y frameworks propuestos, así como una explicación del proceso de plasticidad en su conjunto.

El trabajo de investigación desarrollado se ha guiado por la metodología utilizada tradicionalmente en las Ciencias de la Computación, y que puede resumirse de este modo:

1. Se ha partido del estudio de los marcos conceptuales en los que se enmarca esta tesis. Dentro de la disciplina de la IPO se ha estudiado el campo de las *IUs plásticas*, que ha sido presentado en el *Capítulo 2*, y también el del *enfoque basado en modelos*, presentado en el *Capítulo 3*; por lo que respecta a la disciplina de la Ingeniería del Software el campo de las técnicas se *Separación de Conceptos*, y más concretamente la técnica de la POA, que han sido presentadas en el *Capítulo 6*. Adicionalmente, y de manera tangencial, se ha realizado una incursión en el campo del *groupware*, tratando de descubrir las carencias existentes en el diseño de sistemas específicos para este tipo de entornos.

2. Se han seguido los principios del método científico utilizados en el área de las ciencias básicas para el análisis de la factibilidad de las técnicas de POA para el desarrollo de *sistemas sensibles al contexto*, llevado a cabo en el *Capítulo 6*. Asimismo se ha realizado un extenso estudio de las distintas aproximaciones existentes tanto en el campo de la especificación y desarrollo automática de IUs (*Capítulo 3*), como en el de las arquitecturas del contexto (*Capítulo 5*), a fin de identificar la problemática existente y las limitaciones de las soluciones aportadas.

3. Se han presentado los trabajos relacionados, tanto en el campo de marcos conceptuales de *plasticidad explícita* como en el de aplicación de las técnicas de POA en el campo de la adaptación dinámica (*Capítulo 6*), proporcionando una detallada comparación con respecto a la solución aportada. Este estudio comparativo plasma la parte novedosa de las propuestas presentadas en esta tesis, en especial por el hecho de incorporar las consideraciones de grupo como parte de la información contextual.

4. Por último, se han llevado a cabo *pruebas de concepto* a modo de validación empírica de las expectativas de operación y de calidad de software de la arquitectura propuesta para la parte del cliente. Por lo que respecta al FPE Colaborativo, se ha realizado el ejercicio para el que ha estado concebido: el de estudiar las herramientas existentes basadas en modelo. Se ha podido constatar que el nivel de detalle adquirido al respecto de estas herramientas mediante la aplicación del FPE Colaborativo ha sido más profundo que el que se adquirió inicialmente mediante la aplicación del CAMELEON Reference Framework.

Cabe señalar que gran parte de los resultados obtenidos han sido difundidos a través de la publicación en congresos y revistas científicas, tal y como se detalla en la *sección 9.3*.

9.2. Principales aportaciones

A continuación se recogen las principales aportaciones de la presente tesis.

Por lo que respecta a la infraestructura en su conjunto, las principales aportaciones pueden resumirse en los siguientes puntos:

- Definición y delimitación de los dos niveles de operación que caracterizan un proceso de plasticidad, y que abarcan tanto la fase de diseño como la de ejecución de una IU plástica. La idea es la de aportar el grado apropiado de descentralización y balanceo de carga de responsabilidades entre la operativa del cliente y la del servidor. Este enfoque, denominado *visión dicotómica de plasticidad*, fomenta el estudio por separado de ambas problemáticas, tal y como se plantea inicialmente a través de la *Hipótesis 1*. Hasta ahora tan sólo se había presentado un enfoque monolítico.

- Propuesta de un mecanismo de propagación de cambios contextuales que garantiza la realimentación de la información contextual a ambos lados de la arquitectura cliente-servidor, a fin de producir IUs lo más ajustadas posible a la situación en la que se lleva a cabo la interacción. La validez del mecanismo de comunicación propio de una arquitectura cliente-servidor se plantea inicialmente a través de la *Hipótesis 6*.

- Incorporación de las consideraciones acerca del grupo de trabajo a ambos lados de la arquitectura, con el propósito de que el estado de la situación grupal también tome parte en el proceso de explotación de información contextual, a fin de obtener IUs personalizadas a las circunstancias relativas al grupo.

  El posible beneficio de la relación plasticidad-*awareness* es doble: por un lado ampliar los aspectos de plasticidad, y por otro aportar flexibilidad al diseño *groupware*.

- La modelización del contexto en el nivel de abstracción apropiado, tanto en la parte del cliente como en la parte del servidor, ofreciendo una completa caracterización.

- Un marco conceptual construido bajo el enfoque basado en modelos que estructura el espacio del problema y proporciona las bases para un espacio de solución. Se propone su uso como instrumento de referencia para el estudio de herramientas basadas en modelos, sirviendo además de orientación en el desarrollo de IUs plásticas. Constituye una extensión del CAMELEON Reference Framework.

- Un soporte de software que facilita la construcción y personalización de componentes contextuales y de apoyo al trabajo en grupo, a ser embebidas en la propia operativa del sistema.

A continuación se desarrollan estas dos últimas aportaciones con más detalle.

9.2.1. Marco conceptual de referencia

Por lo que respecta al marco conceptual de referencia propuesto para el nivel de operación propio de la *plasticidad explícita* cabe destacar las siguientes aportaciones:

- Un mecanismo para que ya en la fase de diseño se contribuya a la obtención de adaptaciones proactivas, anticipándose a las necesidades contextuales previsibles en tiempo de ejecución, de manera que los componentes del contexto puedan irse ajustando o cambiando a lo largo de los distintos *ciclos de plasticidad*. De hecho, la posibilidad de anticiparse a los cambios contextuales en la fase de diseño se plantea inicialmente bajo la *Hipótesis 5*.
- Una completa especificación de las reglas, criterios, directivas y bases de conocimiento que participan en el proceso, permitiendo establecer sus dependencias, lo que hace posible un mejor entendimiento del método y del proceso de desarrollo de IUs. Además, la exhaustividad proporcionada otorga la flexibilidad necesaria para la descripción en detalle de cualquier tipo de herramienta basada en modelos.
- Consideración e integración de la tarea en curso que el usuario está llevando a cabo como parte de la descripción del contexto de uso. Se considera una pieza esencial en un enfoque basado en la *visión dicotómica de plasticidad* en general, y en especial en el proceso de derivación de IUs llevado a cabo en el *servidor de plasticidad*.
- Inclusión de heurísticas con el propósito de ofrecer orientación y guía en la especificación y explotación de modelos en el proceso de construcción de la IU, así como la posibilidad de especificar qué componentes y modelos intervienen en el proceso y en qué fase actúan.

Por lo que respecta a la preservación de la usabilidad, aspecto que lo caracteriza como un soporte para el desarrollo de IUs plásticas, y no meramente multi-contextuales, se propone lo siguiente:

- Combinación de las prácticas propias del *diseño centrado en el usuario* con las del *diseño centrado en el uso*, así como el uso de métricas para la evaluación de las propiedades de usabilidad, como por ejemplo las de [CL99], tal y como se realiza en la herramienta AB-UIDE [Lóp05]. Por supuesto, no se descarta la inclusión de diversas directivas de interacción, las cuales recogen la experiencia acumulada por los diseñadores de IUs.

Por lo que respecta a la integración de las consideraciones de grupo en el FPE-Colaborativo, se propone:

- Un esquema conceptual que dispone las directrices y rudimentos necesarios para la incorporación de los aspectos intrínsecos de un entorno colaborativo en una herramienta basada en modelos.

Este conjunto de aportaciones da lugar a un marco conceptual exhaustivo que extiende las propuestas previas.

9.2.2. Soporte para la adaptatividad

La ejecución de *IUs adaptativas*, así como el modo de utilizar la información contextual de forma uniforme continúa planteando grandes retos dentro de la comunidad investigadora. En este sentido se ha propuesto una arquitectura basada en las técnicas de POA que facilita la integración de los mecanismos de adaptación proactiva en cualquier aplicación y plataforma de computación bajo unos cánones de calidad. Los resultados experimentales obtenidos confirman las *Hipótesis 3* y *4*.

En la categoría de arquitecturas del contexto analizadas existen muchos aspectos a mejorar. En particular, la ausencia de un soporte para el almacenamiento del contexto limita el volumen de análisis del contexto que puede ser llevado a cabo por la aplicación. El tipo de contexto manejado en general es limitado, echándose en falta la exploración de técnicas que intenten combinarlos explícitamente con los aspectos relativos al usuario. Por último, los mecanismos internos que soportan la auto-adaptación suelen ser muy específicos del dominio, generando aplicaciones fuertemente acopladas a los componentes del contexto. Dentro de esta área se ha contribuido con:

- El diseño de una arquitectura software que facilita la integración de las capacidades de adaptación en la aplicación subyacente, evitando cualquier impacto sobre la misma y reduciendo el acoplamiento al máximo. Estas propiedades proporcionan reutilización a nivel de diseño.
- Validación empírica de la arquitectura software propuesta, que pone de manifiesto el cumplimiento de las propiedades de calidad de software inicialmente planteadas y prueba su factibilidad en dispositivos compactos. Ello hace pensar en su aplicabilidad en sistemas ubicuos.

Se considera que la problemática relativa a la arquitectura y el almacenamiento del contexto queda resuelta. Puede consultarse una descripción más detallada del análisis de la satisfacción de requisitos identificados por Dey [Dey00] en el uso del FPI en el *Capítulo 8* (véase *Capítulo 8*; *sección 8.3.5.3.*).

Por lo que respecta al aspecto de combinar distintos tipos del contexto, aunque es factible la integración de diversos tratamientos, no se considera explícitamente la implicación conjunta de más de un atributo del contexto. Así por ejemplo, si imaginamos una guía turística para explorar una zona paisajística, las repercusiones que supone un nivel bajo de batería en una zona de campo abierto no son las mismas que cuando el usuario se encuentra en un núcleo de población, o en el interior de un edificio. La combinación de factores como la localización y el nivel de batería puede ser interesante. El carácter eminentemente modular de la arquitectura permite la incorporación de nuevas componentes para tratar la combinación de atributos de este tipo.

Una arquitectura proporciona reutilización a nivel de diseño. Para facilitar la construcción de aplicaciones sensibles al contexto es interesante alcanzar reutilización también a nivel de código. En este sentido destacamos las siguientes aportaciones:

- El diseño de un *framework* de aplicación fácilmente instanciable en componentes del contexto válidas para aplicaciones ya existentes en las que se desee incorporar este tipo de tratamientos.
- Realización de una primera validación de la aplicación del FPI en aplicaciones concretas.
- Un diseño detallado de los componentes del contexto a ser embebidos en la aplicación, en el que se especifica cómo se capturan los datos dependiendo del objetivo de cada caso, qué tipo de datos se manejan y cómo son tratados por el MPI. Por lo tanto, se ha contribuido a proporcionar un medio para expresar información dependiente del contexto en tiempo de ejecución, tal y como se había propuesto en los objetivos iniciales.
- Se proporcionan unas directrices de abstracción y reutilización para ampliar el ámbito de aplicación del framework a distintos dominios, necesidades contextuales o mecanismos de adaptación, concibiéndolo como un framework abierto a futuras extensiones.

A la hora de desarrollar un *framework* cabe plantearse la relevancia y reconocimiento del área de dominio, además de constatarse la ausencia de soluciones similares. Tal y como ha sido analizado, el campo de la *computación sensible al contexto* cumple estos requisitos.

Aunque no existen metodologías de desarrollo de *frameworks* maduras sí existen, sin embargo, guías y principios para su desarrollo. Éstas se recogen en [FHLS98]. Se han procurado y, a nuestro entender, conseguido seguir estas directrices en términos generales. Entre ellas, se han seguido especialmente las siguientes: *"simplifica la interacción entre el framework y las extensiones de la aplicación"*, *"incluye el máximo código posible como parte del framework"*, *"factoriza código y divide extensas abstracciones en otras más pequeñas con el fin de obtener mayor flexibilidad"*.

Uno de los principales beneficios del uso de *frameworks* es la reutilización, aspecto que repercute en un incremento de la productividad, fiabilidad, facilidad de mantenimiento (las aplicaciones desarrolladas a través de un mismo *framework* responden a un diseño y código base común) y calidad del producto resultante (se ofrecen diseños probados que constituyen una base de calidad).

Como propiedades que caracterizan un buen *framework*, existen las siguientes debidas a [Ada95]:

- *Facilidad de uso*. El *framework* debe ser fácil de entender y de usar, de manera que el desarrollo de aplicaciones se vea visiblemente facilitada.
- *Extensibilidad*. Un *framework* es extensible si pueden ser añadidas fácilmente nuevas componentes o propiedades. El que sea extensible hace que sea útil.
- *Flexibilidad*. Habilidad para utilizar el *framework* en más de un contexto.
- *Completitud*. A pesar de que no es posible abarcar todas las variaciones posibles en un determinado dominio, es deseable alcanzar, no obstante, cierta completitud.
- *Consistencia*.

A nuestro entender, el FPI satisface todas estas propiedades. De hecho, la flexibilidad y la extensibilidad son propiedades que lo caracterizan. El aspecto de la consistencia también ha sido especialmente cuidado. La facilidad de uso se ha discutido en detalle en el *Capítulo 8*. A pesar de que el usuario tenga tan sólo leves nociones acerca de las técnicas de POA, una buena documentación y la disponibilidad de aplicaciones de ejemplo puede ser suficiente para su comprensión, haciendo posible su uso de manera sencilla y eficiente.

A pesar de que hace falta comprobar empíricamente la satisfacción de los desarrolladores y el esfuerzo requerido para conocer, comprender y aplicar el *framework* en nuevas aplicaciones, sus características hacen pensar en la posibilidad de ser incluido en el campo del diseño de *aplicaciones sensibles al contexto*.

La versión actual se encuentra en vías de refinamiento y testeo, y por supuesto se requiere completar algunos ciclos más de un proceso iterativo, antes de estar listo para su despliegue. De hecho, existe una regla utilizada entre los desarrolladores de *frameworks* que considera que deben ser desarrolladas tres aplicaciones distintas antes de que un *framework* esté preparado para su distribución.

9.2.3. Aportaciones específicas acerca de la colaboración

La propuesta considera los aspectos relativos al grupo como un parámetro más que enriquece tanto la información contextual como la propia adaptación en el desarrollo de actividades colaborativas.

Aunque existe una amplia variedad de *frameworks* que proporcionan soporte para el desarrollo de aplicaciones groupware, no obstante, la mayoría de ellos no han sido diseñados para soportar la heterogeneidad de dispositivos. En otros casos en los que se incluye este aspecto no se proporciona, sin embargo, un soporte suficientemente dinámico para afrontar la problemática inherente a la movilidad. Por otro lado, el soporte para proporcionar *mecanismos de awareness* no es suficientemente sistemático como para evitar que los desarrolladores tengan que rediseñarlo para cada nuevo sistema. Dentro del campo de groupware se ha contribuido con:

- Un medio para proveer a las aplicaciones de *mecanismos locales de consciencia de grupo*, con el fin de registrar una perspectiva individual acerca de la actividad grupal (la *consciencia de grupo particular*), explotar cualquier interacción entre miembros del grupo de trabajo, así como fomentar nuevas interacciones que puedan ir en favor de la comunicación y la coordinación entre los miembros integrantes. Los componentes de apoyo al grupo son parte de la funcionalidad proporcionada por el FPI.

- Un enfoque para la construcción del *conocimiento compartido*, como resultado de la recopilación de las distintas percepciones individuales acerca del grupo, con el propósito de fomentar la colaboración.

  El despliegue de *IUs sensibles al grupo* permite difundir la denominada *consciencia de conocimiento compartido* [CGPO02], a fin de que el contexto de la actividad grupal pueda ser asimilada por todos los miembros del grupo, y de ese modo trabajar de manera colaborativa.

- Dos niveles de *awareness* consistentes en (a) una perspectiva individual por parte de cada miembro (cliente) que promueve la comunicación y coordinación en tiempo

real; y (b) una perspectiva global (servidor) que ofrece una visión centralizada y común.

En particular, una visión centralizada del *conocimiento compartido* reporta importantes beneficios. Por un lado facilita el mantenimiento de la consistencia entre las distintas percepciones individuales. Asimismo, posibilita la inferencia de propiedades globales en beneficio del grupo. No se ha probado empíricamente, pero se considera que el mantenimiento y combinación de ambas perspectivas favorece la colaboración.

Estos dos niveles de *awareness* se equiparan a los dos niveles de plasticidad introducidos por la *visión dicotómica de plasticidad* persiguiendo, al igual que en el caso de la adaptación, un balance de carga entre cliente y servidor. Este enfoque está en la línea de obtener un compromiso entre el grado de *awareness* y el uso de la red, propuesto por Correa y Marsic en [CM03].

La incorporación y explotación de los aspectos intrínsecos al trabajo en grupo en el propio proceso de plasticidad se planteaba ya inicialmente a través de la *Hipótesis 7*.

9.2.4. Valoración global

La infraestructura de plasticidad propuesta proporciona un enfoque flexible y completo en cuanto a las consideraciones conjuntas de plasticidad y *awareness* que, adicionalmente, otorga un cierto grado de sistematización en el diseño, desarrollo y ejecución de IUs plásticas y *sensibles al grupo* dinámicamente reconfigurables, tratando de evitar en la medida de lo posible *discontinuidades de plasticidad*.

El hecho de que esta infraestructura proporcione un medio para compartir el contexto, así como la posibilidad de poder entregar a los miembros del grupo IUs especialmente personalizadas a cada situación grupal, como mecanismo para difundir la *consciencia de conocimiento compartido*, son las dos argumentaciones principales a favor de su adecuación para entornos colaborativos.

El enfoque basado en la *visión dicotómica*, fundamentado en el propósito de alcanzar un equilibrio operacional entre cliente y servidor, su factibilidad para entornos colaborativos y el hecho de que los componentes de adaptación sean compactos y soportables por dispositivos de capacidades limitadas, hacen pensar en su validez para entornos ubicuos y pervasivos.

Se puede concluir que se han aportado artefactos software de utilidad tanto para el diseño como para la ejecución de IUs plásticas, los cuales han estado especialmente ideados

bajo el enfoque de la *visión dicotómica*. La explicación detallada del proceso global de plasticidad, así como la visión integradora de la operativa conjunta presentada en el *Capítulo 8* muestra el engranaje de todas las piezas, poniendo de manifiesto la complementariedad de ambos niveles de operación (véase *Capítulo 8*; *sección 8.4.*).

9.3. Publicaciones realizadas

Parte del trabajo realizado durante esta tesis ha sido publicado en los siguientes capítulos de libro:

- Sendín, M.; Lorés, J. "Local Support to Plastic User Interfaces: an Orthogonal Approach". *Selection of HCI related papers of Interacción 2004*. Navarro-Prieto & Lorés Eds. Computer Science Series. Springer-Verlag (2005) pp. 163-168. ISBN: 978-1-4020-4202-1.
- Sendín, M.; Lorés, J. "Remote Support to Plastic User Interfaces: a Semantic View". *Selection of HCI related papers of Interacción 2004*. Navarro-Prieto & Lorés Eds. Computer Science Series. Springer-Verlag (2005) pp. 55-70. ISBN: 978-1-4020-4202-1.

Se ha publicado en diversos Congresos Nacionales e Internacionales. A continuación se presentan las publicaciones más relevantes clasificadas de acuerdo al foco de atención en que se centra cada una de ellas.

Una descripción de la infraestructura software en su conjunto puede encontrarse en:

- Sendín, M.; Lorés, J.; Solà, J. "Towards the tailoring of a ubiquitous interactive model applied to the natural and cultural heritage of the Montsec Area". *Proceeding of the workshop Personalization techniques in electronic publishing on the web: Trands and perspectives*, en conjunción con el 2nd International conference of Adaptative Hypermedia and adaptative web based systems (AH 2002) ISBN: 699-8193-5 (2002) pp. 57-69.
- Sendín, M.; Lorés, J. "Plasticity in Mobile Devices: a Dichotomic and Semantic View". *Proceedings of Workshop on Engineering Adaptive Web*, en conjunción con Adaptive Hypermedia (AH 2004). ISSN: 0926-4515 (2004) pp. 58-67.
- Sendín, M.; Lorés, J. "Dichotomy in the Plasticity Process: Architectural Framework Proposal". *Actas del V Congreso en Interacción Persona-Ordenador (Interacción 2004)*. ISBN: 84-609-1266-3 (2004).
- Sendín, M. "Infrastructure for Plastic User Interfaces under a Dichotomic View". *Adjunct Proceedings of Communicating Naturally Through Computers (Interact 2005)* (2005) pp. 57-58.

La arquitectura software propuesta para el *Motor de Plasticidad Implícita* se describe en:

- Sendín, M.; Lorés, J. "Plasticidad implícita en Dispositivos Móviles: Hacia la Ortogonalidad deseada en la Separación de Conceptos". *Actas del V Congreso en Interacción Persona-Ordenador (Interacción 2004)*. ISBN: 84-609-1266-3 (2004) pp- 70-78.
- Sendín, M.; Lorés, J. "Towards the Design of a Client-Side Framework for Plastic UIs using Aspects". *Proceedings of the International Workshop on Plastic Services for Mobile Devices (PSMD05)*, en conjunción con el Interact 2005, Communicating Naturally Through Computers (2005).
- Sendín, M.; Lorés, J. "Putting Aspects to Work in the Design of a Local Framework for Plastic UIs". *Actas del VI Congreso en Interacción Persona-Ordenador*, en conjunción con el I Congreso Español De Informática (CEDI 2005) (2005).

El enfoque para el marco conceptual inicial (sin componentes para el groupware), esto es, el *Framework de Plasticidad Explícita*, se encuentra descrito en:

- Sendín, M.; Lorés, J. "Interfaces de Usuario Plásticas: Diseñando para el Cambio". *Actas del Taller de Sistemas Hipermedia Colaborativos y Adaptativos*, en conjunción con las VIII Jornadas de Ingeniería del Software y Bases de Datos. (2003).
- Sendín, M.; Lorés, J. "Plastic User Interfaces: Designing for Change". *Proceedings of the Making model-based UI design practical: usable and open methods and tools Workshop*. Joint Conference of Intelligent User Interfaces 2004 and Computer Aided and Design of User Interfaces 2004. Hallvard Trætteberg, Pedro J. Molina, Nuno J. Nunes Eds. CEUR-Workshop Proceedings. ISSN: 1613-0073. Vol. 103 (2004) On-line: http://CEUR-WS.org/Vol-103
- Sendín, M.; Lorés, J. "Hacia un Motor de Plasticidad Explícita Retroalimentado". *Actas del V Congreso en Interacción Persona-Ordenador (Interacción 2004)*. ISBN: 84-609-1266-3 (2004).
- Sendín, M.; Lorés, J.; Montero, F.; López-Jaquero, V. "Towards a Framework to Develop Plastic User Interfaces". Lecture Notes on Computer Science Series. *Proceedings of Human-Computer Interaction with Mobile Devices and Services (MobileHCI 2003)*. Luca Chittaro Eds. Springer-Verlag. Vol. 2795. ISSN: 0302-9743. ISBN:3-540-40821-5 (2003) pp. 428-433.
- Sendín, M.; Lorés, J.; Montero, F.; López-Jaquero, V.; Gonzàlez P. "User Interfaces: A Proposal for Automatic Adaptation". Lecture Notes in Computer Science Series. *Proceedings of International Conference on Web Engineering (ICWE 2003)*. Springer-Verlag. Vol. 2722. ISBN: 3-540-40522-4 (2003) pp. 108-113.
- Sendín, M.; Lorés, J.; Montero, F.; López-Jaquero, V. "Using Reflexion on Dynamic Adaptation of User Interfaces". *Proceedings of the 4th International Workshop on Mobile Computing (IMC 2003)*. ISBN: 3-8167-6319-7 (2003).

Como publicaciones que presentan las técnicas de plasticidad adaptadas a la problemática de groupware existen dos: una centrada en la infraestructura del cliente y la otra en la del servidor.

- Sendín, M.; Collazos, C. A.; López-Jaquero, V. "Framework de Plasticidad Explícita: Un Marco Conceptual de Generación de IUs Plásticas para entornos Colaborativos". *Actas del VII Congreso en Interacción Persona-Ordenador (Interacción 2006)* (2006).

- Sendín, M.; Collazos, C.A. "Implicit Plasticity Framework: A Client-Side Generic Framework for Collaborative Activities". Lecture Notes on Computer Science Series. *Proceedings of the 12th International Workshop on Groupware (CRIWG 2006).* Springer-Verlag Vol. 4154 (2006) pp. 219-227.

Una primera aproximación de la infraestructura de plasticidad propuesta aplicada al campo de la Computación Ubicua se puede encontrar en:

- Sendín, M. "Dichotomy in the Plasticity Process: Architectural Framework Proposal". A. Ferscha, H. Hörtner, G. Kotsis Eds. *Advances in Pervasive Computing.* Oesterreichische Computer Gesellschaft. Austrian Computer Society. Doctoral Colloquium in Pervasive Computing. ISBN:3-85403-176-9 (2004) pp. 141-147.
- Sendín, M. "Implicit Plasticity Framework: a Client-Side Generic Framework for Context-Awareness". *International Conference on Ubiquitous Computing (ICUC'06).* J.R. Hilera, J. A. Gutiérrez de Mesa, R. Barchino, J. M. Gutiérrez Eds. CEUR-Workshop Proceedings. ISSN: 1613-0073. Vol. 208 (2006) On-line: http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-208/.

También ha sido presentado en dos ocasiones en un congreso específico en la temática de desarrollo de software *orientado a aspectos*. Estas son las dos publicaciones:

- Sendín, M.; Lorés, J. "Aspectual Decomposition in the Design of a Framework for Context-Aware User Interfaces". *Actas del Taller de Desarrollo de Software Orientado a Aspectos (DSOA 2005)*, en conjunción con las X Jornadas de Ingeniería del Software y Bases de Datos (2005).
- Sendín, M.; Viladrich, J. "Contrasting aspectual decompositions with object-oriented designs in contexts-aware mobile applications". *Actas del Taller de Desarrollo de Software Orientado a Aspectos (DSOA 2006)*, en conjunción con las XI Jornadas de Ingeniería del Software y Bases de Datos (2006).

Publicaciones en revistas:

- Sendín, M.; Lorés, J.; Aguiló, C.; Balaguer, A. "A ubiquitous interactive computing model applied to the natural and cultural heritage of the Montsec area". *Upgrade. The European online magazine for the information technologies professional.* Vol. 2, núm. 5 (2001) http://www.upgrade-cepis.org/issues/2001/5/upgrade-vII-5.html.
- Sendín, M.; Lorés, J.; Aguiló, C.; Balaguer, A. "Un modelo interactivo ubicuo aplicado al patrimonio natural y cultural del área del Montsec". *Revista Iberoamericana de Inteligencia Artificial (AEPIA).* ISSN: 1137-3601. Vol. 16 (2002) pp. 43 - 48.

Además, se encuentra en proceso final de evaluación para una revista indexada en Science Citation Index el siguiente trabajo.

- Sendín, M.; López-Jaquero, V.; Collazos, C.A. "Collaborative Explicit Plasticity Framework: a Conceptual Scheme for the Generation of Plastic and Group-Aware User Interfaces". *Journal of Universal Computer Science.*"Designing the Human-Computer Interaction: Trends and Challenges" - special issue presenting a selection of the best papers from the "Human-Computer Interaction 2006" conference. C. Bravo, M. A. Redondo & M. Ortega (Eds.) (Diciembre 2007) *En prensa.*

9.4. Trabajo Futuro

Los desarrollos obtenidos en esta tesis dejan el camino abierto a nuevas líneas de trabajo que complementan el ya realizado. No obstante, es primordial completar etapas de testeo y formalización, así como la realización de otro tipo de pruebas de concepto relacionadas con la infraestructura de plasticidad propuesta.

9.4.1. Trabajo inmediato

En lo que respecta a la aplicabilidad de la infraestructura en entornos colaborativos, se tiene planeado lo siguiente:

A) Acabar de formalizar el lenguaje basado en XML a utilizar para enviar las peticiones de adaptación al *servidor de plasticidad*, en el que también se incluya la descripción de situaciones de grupo. Aunque este tipo de cuestiones han formado parte integrante de los experimentos realizados, no obstante, han consistido en algunas pruebas del concepto. Debe diseñarse y definirse una estructura para este tipo de documentos XML que contemple ampliamente todas las consideraciones relativas a la situación contextual.

B) Resolver la automatización y puesta en funcionamiento en la entrega de *midlets* a la plataforma cliente. Actualmente el soporte existente para lanzar automáticamente midlets sin necesidad de ser inicializados por el usuario es a través de la clase *PushRegistry*[2], que hace posible su activación por conexiones de red entrantes. Por otro lado, la nueva especificación MIDP 3.0 promete ciertas aportaciones en el terreno de midlets auto-arrancables.

C) Poner en práctica la base teórica relacionada con el tratamiento de la información de *awareness* ideada y plasmada en el *FPE Colaborativo* en una herramienta basada en modelos concreta. La herramienta de la que se dispone es AB-UIDE, que dará lugar a la primera implementación del marco conceptual. Para ello se requiere formalizar el modelado del *conocimiento compartido.*

Una vez implementada la versión colaborativa de AB-UIDE se planea la realización de pruebas en entornos colaborativos reales, como pudiera ser a través de la aplicación

[2]del paquete *javax.microedition.io*, incorporada en la versión MIDP 2.0.

colaborativa utilizada en el primer caso de estudio descrito en el *Capítulo 7*. La intención es la de validar y analizar de alguna manera si el soporte conjunto (parte *explícita* e *implícita*) facilita la colaboración, y de algún modo valorar en qué medida.

Adicionalmente, este esfuerzo servirá para evaluar cuantitativamente la productividad, escalabilidad y flexibilidad de la propuesta en su conjunto.

D) Por otro lado, y tal y como ya se ha mencionado, aunque se han realizado pruebas de concepto sobre aplicaciones concretas, éstas no son suficientes para alcanzar el nivel de refinamiento y de test necesario para poder considerar el FPI listo para su despliegue.

Es por ello que se planea la realización de nuevos ciclos de refinamiento incluyendo también aplicaciones de sobremesa. Además, nos interesa analizar el uso del framework por parte de distintos desarrolladores, a fin de valorar su facilidad de uso, así como su grado de aplicabilidad al utilizarlo en distintos dominios de interés.

En este sentido, en estos momentos disponemos de una aplicación que recrea el clásico juego de consola conocido como 'Arkanoid', desarrollado también en J2ME. Los *midlets* destinados a juegos utilizan una API denominada de bajo nivel para la construcción de la IU, la cual proporciona un control mucho mayor sobre los recursos de la IU, así como la posibilidad de capturar eventos de bajo nivel. Además, este campo da mucho juego tanto para añadir funcionalidades extra de las que sacar partido de la potencialidad del dispositivo -posibilidad de jugar en red, uso de mensajería u otras extensiones-, como para mejorar la IU, incorporando aspectos de sonido, vibración, imágenes u otro tipo de elementos que hagan la IU más llamativa. Es con esta aplicación que se tiene en mente experimentar la parte de adaptación a los recursos hardware tanto de tipo estático -tamaño de pantalla, tipo de mensajería, posibilidades de conexión, etc.-, como de tipo dinámico -disponibilidad de memoria, nivel de batería o estado de la red.

Esta fase de pruebas, juntamente con la elaboración de una adecuada documentación concluiría los pasos requeridos previo a su despliegue como librería jerárquica.

9.4.2. Líneas Futuras

Por lo que respecta a líneas futuras se han planteado las siguientes:

- Incorporar mecanismos para facilitar la toma de decisiones acerca de qué dispositivos o plataformas de computación son las más apropiadas para los distintos escenarios de trabajo de acuerdo a distintos factores, entre los cuales se encuentran las condiciones de trabajo en grupo, si es el caso. Lo ideal sería que el usuario pudiera escoger si desea recibir sugerencias por parte del sistema.

Con este propósito, enfocado concretamente en el campo de entornos colaborativos, se cuenta con el trabajo de Alarcón et al. en [AGOP05], donde se propone un framework de diseño que pretende ofrecer una herramienta útil para identificar los escenarios colaborativos para los que es favorable la utilización de dispositivos móviles. Se materializa en la distinción de hasta seis contextos que deben ser tenidos en cuenta en la fase de diseño de este tipo de aplicaciones. En un trabajo posterior Guerrero et al. dan un paso más allá tratando de identificar los tipos de dispositivos que resultan apropiados para soportar colaboración móvil en contextos de trabajo específicos [GOPC06].

- Contemplar el rol del usuario y su evolución implícita en la representación de la *consciencia de grupo particular*, a fin de incorporar las posibles repercusiones de su evolución, así como la personalización de acciones relacionadas con el grupo de acuerdo a este parámetro.

- Incorporar otras técnicas existentes en el campo de CSCW para estimular la participación de los miembros del grupo, tales como las de Vassileva en [VCSH04].

- Incorporar en la infraestructura propuesta algunos mecanismos de visualización de la información de *awareness*, tanto de la *consciencia de grupo particular* como del *conocimiento compartido*. En particular, técnicas de este tipo son especialmente interesantes en el campo de CSCL (Computer Supported Collaborative Learning)[3], donde la construcción y mantenimiento de un *conocimiento compartido* del problema resulta especialmente necesario para lograr el éxito del trabajo colaborativo. En este sentido, las técnicas de visualización pueden resultar de gran ayuda.
  Por lo que respecta a este tema cabe destacar el trabajo de Gutwin y Greenberg. En [GG02] presentan su teoría descriptiva para soportar el diseño de *groupware*, enfocada en una clase particular de *awareness* denominada *workspace awareness*, y materializada a través de un framework. Muestran las distintas formas en que los diseñadores de *groupware* pueden aplicar el conocimiento recogido en el framework para el diseño de este tipo de IUs, considerando aspectos como la representación y disposición de la información de *awareness* en la IU. Además, proponen estrategias para el diseño de los denominados *awareness widgets*.

- La continua proliferación de dispositivos con acceso a Internet plantea la necesidad de adaptar no sólo el contenido Web a las distintas audiencias -objetivo por excelencia de la *Hipermedia Adaptativa*-, sino también la IU a las distintas características físicas de la plataforma de interacción. Se hace indispensable conjuntar ambos esfuer-

[3]Un tipo particular de sistemas CSCW focalizado en objetivos pedagógicos y de aprendizaje.

zos para proporcionar unos contenidos personalizados y ajustados a las restricciones de espacio y modo de interacción con los que se cuente.
Una primera aproximación al respecto se describe en [SMC+07], donde se presenta una arquitectura en la que se integran los modelos SEM-HP [MMG05] y AB-UIDE [Lóp05] con objeto de personalizar y adaptar de forma dinámica el contenido y la IU de una aplicación Web. En esta arquitectura se propone igualmente una descomposición en diversos niveles de concreción, con el fin de separar las partes o capas de software que exhiben diferentes tipos o grados de adaptación, estableciendo distintos niveles de abstracción.

- Identificar algún tipo de directiva de usabilidad en la que se tenga en cuenta explícitamente el trabajo colaborativo, a fin de ser integrada en el *FPE Colaborativo*.

  En este sentido, el trabajo desarrollado en el grupo GRIHO [GCG06] constituye una primera aproximación dentro de esta línea, en la que se proponen una serie de guías que tienden a asegurar un correcto nivel de *consciencia de conocimiento compartido* en sistemas CSCL.

- Proporcionar un enfoque semántico al *Framework de Plasticidad Explícita*, con el propósito de dotarlo de la suficiente flexibilidad como para poder considerar múltiples parámetros, siguiendo los principios del *diseño universal* [FVB+01]. Este aspecto permitiría mejorar el aspecto de la flexibilidad en la adaptación de contenidos en este tipo de técnicas *basadas en modelos*.
  Se trataría de incorporar una capa conceptual en la que, a través de una ontología, los expertos del dominio definieran conceptos comunes, relaciones y atributos, para posteriormente ser instanciados en cada contexto de uso correspondiente a un dominio. Eso permitiría obtener tanto el conjunto de modelos apropiado como la información a almacenar, con objeto de derivar el *modelo conceptual* de manera flexible, el cual quedaría definido en base a cierta información semántica.
  Un campo de aplicación por excelencia sería el del e-Learning, con objeto de proporcionar los beneficios del extenso y diversificado conocimiento contenido en las actuales librerías digitales.

  En particular, estas dos últimas propuestas se plantean como líneas integradoras del esfuerzo de investigación que se está llevando a cabo por parte de varios miembros del grupo GRIHO.

- Por último, y como línea que personalmente me resulta más estimulante, es la de revisar y especializar la infraestructura propuesta para la construcción de sistemas colaborativos especialmente ideados para entornos ubicuos y de inteligencia ambiental.

Existen dos entornos por excelencia utilizados como fuente de inspiración para idear y diseñar escenarios de *computación ubicua*, y en los que unas adecuadas técnicas de comunicación y coordinación pueden ser de gran ayuda para alcanzar colaboración. Se trata de los entornos de hospitales -destacando el trabajo del grupo de Favela [FRPG04]- y el del M-Learning -en el que destacan los proyectos del grupo CHICO[4], tales como e-CLUB, y su continuación en el proyecto AULA (AULA_IE y AULA_T).

[4]Computer-Human Interaction and Collaboration (Universidad de Castilla-La Mancha). On-line: http://chico.inf-cr.uclm.es/webchico/

Apéndice A

Visión general del lenguaje *AspectJ*

En este apéndice se describe brevemente el subconjunto de la sintaxis del lenguaje AspectJ manejado en la construcción del FPI, ofreciendo una visión general del lenguaje. Se trata de una referencia incompleta que puede ser ampliada en la Web oficial: *AspectJ project*[1], que provee documentación actualizada del lenguaje, así como en diversos libros, como por ejemplo [GL03] y [Lad03].

[1] http://www.eclipse.org/aspectj/.

A.1. Introducción

AspectJ constituye una de las primeras implementaciones del paradigma *orientado a aspectos*. Es un lenguaje relativamente nuevo y por lo tanto en continua evolución. Consiste en una extensión de Java para soportar la definición y manejo de *aspectos*, por lo que cualquier programa válido para Java, lo será también para *AspectJ*, ya que el compilador de *AspectJ* genera *bytecodes* compatibles con cualquier JVM.

Precisamente, el que *AspectJ* sea una extensión de Java hace que el lenguaje sea fácil de aprender, puesto que tan sólo añade un conjunto de construcciones al lenguaje base, conservando las características ventajosas de Java, como por ejemplo la independencia de la plataforma. Para ser más exactos, la funcionalidad de los *aspectos* se expresa utilizando Java estándar, mientras que las *reglas de composición* (reglas que especifican la manera como realizar la composición de todos los conceptos implementados independientemente -véase *Capítulo 6*; *sección 6.2.3.3.2*-) se especifican mediante una serie de construcciones proporcionadas por el propio lenguaje.

La manera como especificar las *reglas de composición* (su sintaxis y sus posibilidades) es lo que caracteriza la esencia de cada lenguaje de Programación Orientada a Aspectos (POA). *AspectJ* brinda dos formas de implementación transversal. Son las siguientes:

- *Implementación transversal dinámica.* Permite definir comportamiento adicional (afectando, modificando o añadiendo código), el cual se ejecutará en puntos específicos dentro del programa.

  Ésta se especifica a través de los *puntos de enlace*, *puntos de corte* y *consejos* (véanse definiciones en el *Capítulo 6*; *sección 6.2.3.3.3*).

- *Implementación transversal estática.* Permite modificar la estructura estática del programa, agregando superclases o implementaciones de interfaces. La función principal de ésta es dar soporte a la *implementación transversal dinámica.*

  Existen varias construcciones para especificar la *implementación transversal estática.* Tan sólo se hará mención aquí de las declaraciones inter-tipo, las declaraciones de parentesco y las declaraciones de precedencia, al ser las que son utilizadas directamente en la implementación del FPI.

En cualquier caso la modificación del comportamiento o de la estructura de clases se realiza una unidad de programa localizada y modular de tipo *aspecto.*

A.2. Implementación transversal dinámica

Se trata del tipo de *implementación transversal* más utilizada en *AspectJ*.

Conviene recordar aquí los conceptos asociados.

- **Puntos de enlace** (*joinpoint*): son puntos bien definidos en la ejecución de un programa. Por ejemplo, la llamada a un método, el acceso a un atributo, etc. Representan los puntos del programa donde los *aspectos* añaden su comportamiento.
- **Puntos de corte** (*pointcut*): agrupan *puntos de enlace* para los que se requiere aplicar un mismo comportamiento o tratamiento extra y permiten exponer su contexto a los *consejos*.
- **Consejos** (*advice*): especifican el código que se ejecutará en los *puntos de enlace* que satisfacen cierto *punto de corte*, pudiendo acceder al contexto de dichos *puntos de enlace*.

Así, los *puntos de corte* indican el '*dónde*' y los avisos el '*qué*', apareciendo siempre asociados cada *punto de corte* con un *consejo*. Un ejemplo de utilización de este tipo de implementación transversal sería añadir cierto comportamiento tras la ejecución de ciertos métodos o sustituir la ejecución normal de un método por una ejecución alternativa.

A.2.1. Pointcuts

La construcción *pointcut* es propia del lenguaje *AspectJ* y, a diferencia de los *advices* y los *aspectos*, no se equipara a ninguna construcción *orientada a objetos* de Java asociada a la construcción de clase.

A.2.1.1. Sintaxis

Un *pointcut* puede tener un nombre asignado, facilitando así su reutilización. En caso contrario se les denomina anónimos, y deben ser declarados junto al *advice* con el que están asociados.

La sintaxis de un *pointcut* con nombre es la siguiente:

```
<pointcut> ::= <access_type> [abstract] pointcut
<pointcut_name>({<parameters>}) : {[!] designator [ && | || ]};
designator ::= designator_identifier(<signature>)
```

Consta del modificador de acceso (*access_type*) que, por defecto es *package*, la palabra reservada *pointcut* seguida del nombre del *punto de corte* y de sus parámetros -o lista vacía si no los hay-, que permiten exponer el contexto de los *joipoints* a los *advice*s. Por último, vienen las expresiones que identifican los *joinpoints* precedidas del carácter dos puntos "**:**".

Opcionalmente, los *pointcuts* pueden ser abstractos, caso en el que su especificación queda delegada en los *sub-aspectos* (mecanismos de herencia).

La sintaxis de un *pointcut* anónimo es la siguiente:

```
<pointcut> ::= { [!] designator [ && | || ] };
```

Consta de una o más expresiones (*designator*) que identifican una serie de *joinpoints*. Cada una de estas expresiones está formada por un descriptor de *pointcut*, que especifica su tipo, y una signatura, que indica el punto de la ejecución del programa donde se aplica, es decir, el *joinpoint*.

Tal y como aparece reflejado, se pueden combinar las expresiones que identifican los *joinpoints* mediante los operadores lógicos ! (NOT), | | (OR) y && (AND). Así, la expresión compuesta ***exp1*** | | ***exp2*** se cumple cuando se satisface al menos una de las dos expresiones, *exp1* o *exp2*. La expresión compuesta ***exp1 && exp2*** se cumple cuando se cumple *exp1* y *exp2*. Y la expresión **!*exp*** se cumple cuando no se satisface *exp*.

La figura A.1 muestra la estructura de un *pointcut* con nombre. Como se observa, la especificación de los *joinpoints* se introduce a continuación de los dos puntos. El ejemplo incluido muestra uno de los *pointcuts* utilizados en el *aspecto Colaboración* (componente de apoyo al trabajo en grupo para J2ME -véase *Capítulo 8*-).

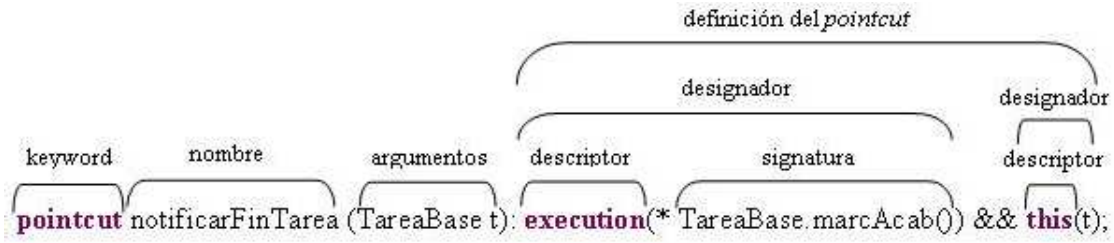


Figura A.1: Definición de un *pointcut* con nombre.

Consta de dos *joinpoints* combinados con el operador lógico AND. Las palabras clave *execution* y *this* son tipos de *pointcuts* –se presentan a continuación-, y ** TareaBase.marcAcab()* es la signatura que hace referencia al método *marcAcab* de la clase *TareaBase*.

A.2.1.1.1. Comodines

A la hora de expresar los *joinpoints* se pueden usar una serie de comodines para identificar diversos *joinpoints* que tienen características comunes, como por ejemplo la signatura. El significado de estos comodines dependerá del contexto en el que aparezcan:

***** : la interpretación para el asterisco depende del contexto en el que aparezca. En algunos contextos significa cualquier número de caracteres excepto el punto. En otros representa cualquier tipo (clase, interfaz, tipo primitivo o aspecto). En el caso del ejemplo anterior significa cualquier modificador de acceso.

.. : el carácter dos puntos representa cualquier número de caracteres, incluido el punto. Cuando se usa para indicar los parámetros de un método, significa que el método puede tener un número y tipo de parámetros arbitrario.

+ : el operador suma representa una clase y todos sus descendientes, tanto directos como indirectos.

La precedencia de estos operadores lógicos es la misma que en Java. Es recomendable el uso de paréntesis para hacer más legible las expresiones o para modificar la precedencia por defecto de los operadores.

A.2.1.2. Tipos de pointcuts

AspectJ proporciona una serie de descriptores de *pointcuts* que permiten identificar grupos de *joinpoints* que cumplen diferentes criterios. Estos descriptores se clasifican en los distintos grupos, los cuales se desarrollan a continuación.

A.2.1.2.1. Pointcuts basados en las categorías de Joinpoint

Este tipo de puntos de corte captura los *joinpoints* según la categoría a la que pertenecen. Existe un descriptor para cada categoría. La tabla A.1 recoge la sintaxis de cada uno de ellos.

Los descriptores *call* y *execution* son, de manera destacada, los más utilizados. Es importante distinguirlos bien, puesto que a menudo se confunde su semántica. Los *pointcuts* de tipo *call* permiten introducir el comportamiento en el contexto de la invocación de un método, y por lo tanto, el contexto que se captura es el del objeto que envía el mensaje. Por el contrario, los *pointcuts* de tipo *execution* referencian al contexto de ejecución del método invocado. Esto es, el contexto es del receptor del mensaje.

Descriptor	**Categoría *Joinpoint***
call(methodSignature)	Llamada a método
execution(methodSignature)	Ejecución de método
call(constructorSignature)	Llamada a constructor
execute(constructorSignature)	Ejecución de constructor
get(fieldSignature)	Lectura de atributo
set(fieldSignature)	Asignación de atributo
handler(typeSignature)	Ejecución de manejador
staticinitialization(typeSignature)	Inicialización de clase
initialization(constructorSignature)	Inicialización de objeto
preinitialization(constructorSignature)	Pre-inicialización de objeto
adviceexecution()	Ejecución de aviso

Cuadro A.1: Sintaxis de los *pointcuts* según la categoría de los *joinpoints*.

El parámetro de estos *pointcuts* es, en general, la signatura de un método, tal y como se observa en la figura A.1. En ese caso el *pointcut* captura la llamada al método *marcAcab* en la clase *TareaBase*, que no tiene parámetros ni valor de retorno. No se restringe el modificador de acceso. Si cualquiera de estas condiciones no se cumpliera, el *pointcut* no capturaría ningún método.

La Tabla A.2 muestra algunos ejemplos de uso de este tipo de puntos de corte:

Pointcuts	***Joinpoints* identificados**
call(void Punto.set(..))*	Llamadas a los métodos de la clase Punto, cuyo nombre empieza por *set*, devuelven *void* y tienen un número y de parámetros arbitrario.
execute(public Elemento.new(..))	Ejecución de todos los constructores de la clase *Elemento*
*get(private * Linea.*)*	Lecturas de los atributos privados de la clase *Linea*.
set(Punto.*)*	Escritura de cualquier atributo de la clase *Punto*.
handler(IOException)	Ejecución de los manejadores de la excepción *IOException*
initialization(Elemento.new(..))	Inicialización de los objetos de la clase *Elemento*
staticinitialization(Logger)	Inicialización estática de la clase *Logger*
adviceexecution() && within(MiAspecto)	Todas las ejecuciones de *advices* pertenecientes al *aspecto MiAspecto*. No se puede capturar la ejecución de un único *advice*.

Cuadro A.2: Ejemplos de *pointcuts* basados en la categoria de los *joinpoints*.

A.2.1.2.2. Pointcuts basados en flujo de control

Este tipo de *pointcut* captura *joinpoints* de cualquier categoría, siempre y cuando ocurran en el contexto de otro *punto de corte*. Los descriptores son *cflow* y *cflowbelow*.

La Tabla A.3 recoge la sintaxis y la semántica de los descriptores basados en flujo de control.

Sintaxis	**Semántica**
cflow(pointcut)	Captura todos los *joinpoints* en el flujo de control del *pointcut* que se le pasa como parámetro, incluidos los *joinpoints* identificados por el *pointcut*.
cflowbelow(pointcut)	Captura todos los *joinpoints* en el flujo de control del *pointcut* que se le pasa como parámetro, excluidos los *joinpoints* identificados por el *pointcut*.

Cuadro A.3: Sintaxis y semántica de los *pointcuts* basados en flujo de control.

Para una mejor comprensión de este tipo de *pointcuts* se muestran a continuación una serie de ejemplos ilustrativos.

Pointcut	***Joinpoints* identificados**
cflow(call(Cuenta.debito (..)))*	Todos los *joinpoints* en el flujo de control del método *debito* incluida la propia llamada al método *debito*. El *pointcut* que se pasa como parámetro es anónimo.
cflowbelow(call(Cuenta. debito(..)))*	Todos los *joinpoints* en el flujo de control del método *debito*, sin incluir la llamada al método *debito*. El *pointcut* que se pasa como parámetro es anónimo.
cflow(operaciones())	Todos los *joinpoints* en el flujo de control de los *joinpoints* capturados por el *pointcut* con nombre *operaciones*
cflowbelow(execution(Cuenta. new(..)))	Todos los *joinpoints* durante la ejecución de cualquier constructor de la clase *Cuenta*, sin incluir la propia ejecución del método.

Cuadro A.4: Ejemplos de *pointcuts* basados en flujo de control.

Un uso típico de estos descriptores es cuando se trabaja con métodos recursivos. Por ejemplo, supongamos que tenemos un método recursivo llamado *factorial(int)* dentro de la clase *Math*. El siguiente *pointcut* identifica todas las llamadas al método recursivo.

```
pointcut llamadas(): call(int Math.factorial(int));
```

Mientras que éste otro punto de corte sólo capturará la primera llamada al método recursivo:

```
pointcut primera(): llamadas() && !cflowbelow(llamadas());
```

A.2.1.2.3. Pointcuts basados en localización de código

Este tipo de *pointcut* captura *joinpoints* de cualquier categoría que se localizan en ciertos fragmentos de código, por ejemplo, dentro de una clase o dentro del cuerpo de un método. Estos descriptores son *within* y *withincode*.

La Tabla A.5 recoge la sintaxis y la semántica de los descriptores de estos *pointcuts*.

Sintaxis	**Semántica**
within(typeSignature)	Captura todos los *joinpoints* dentro del cuerpo de las clases o *aspectos* especificados mediante *typeSignature*
withincode(methodSignature) *withincode(constructorSignatu-re)*	Captura todos los *joinpoints* dentro del cuerpo de los métodos o constructores especificados por *methodSignature* y *constructorSignature* respectivamente.

Cuadro A.5: Sintaxis y semántica de los *pointcuts* basados en localización.

En la siguiente tabla se muestran ejemplos de este tipo de *pointcut*.

Pointcut	***Joinpoints* identificados**
within(Elemento)	Todos los *joinpoints* dentro de la clase *Elemento*.
within(Elemento+)	Todos los *joinpoints* dentro de la clase *Elemento* y sus subclases.
withincode(Punto.set*(..))*	Todos los *joinpoints* dentro del cuerpo de los métodos de la clase *Punto* cuyo nombre empieza por *set*.

Cuadro A.6: Ejemplos de *pointcuts* basados en localización de código.

Un uso común del descriptor *within* es excluir los *joinpoints* generados por el propio *aspecto*. Por ejemplo, el siguiente *pointcut* es utilizado en el *aspecto Colaboración* (componente de apoyo al trabajo en grupo para J2ME -véase *Capítulo 8*-) para capturar las invocaciones al método *CancelarT*, siempre que no hayan estado realizadas desde el propio *aspecto*.

```
pointcut comunicarTareaCancelada(TareaBase t): call(* Tarea.CancelarT()) && target(t)
                                                && ! within(ColaboracionJ2ME);
```

A.2.1.2.4. Pointcuts basados en objetos

Este tipo de *pointcut* captura los *joinpoints* cuyo objeto actual (*this*) u objeto destino (*target*) en tiempo de ejecución son de un cierto tipo. Entendemos por objeto actual el objeto que se está ejecutando actualmente y por objeto destino, el objeto sobre el que se invoca un método.

Además, permite exponer el contexto de los *joinpoints*. Los descriptores de puntos de corte basados en objetos son:

Sintaxis	**Semántica**
this(typeSignature)	Captura todos los *joinpoints* cuyo objeto actual es una instancia del tipo especificado por *typeSignature* o cualquiera de sus subclases.
this(<identifier>)	Expone el objeto actual del *joinpoint* y lo guarda en la variable de nombre *<identifier>*
target(typeSignature)	Captura todos los *joinpoints* cuyo objeto destino es una instancia del tipo especificado por *typeSignature* o cualquiera de sus subclases.
target(<identifier>)	Expone el objeto destino del *joinpoint* y lo guarda en la variable de nombre *<identifier>*

Cuadro A.7: Sintaxis y semántica de los *pointcuts* basados en objetos.

La expresión *typeSignature* en los descriptores *this* y *target* no puede contener los comodines asterisco y dos puntos. El comodín +, no se necesita porque las subclases ya son tenidas en cuenta por las reglas de herencia de Java.

La Tabla A.8 muestra algunos ejemplos de uso de este tipo de *pointcuts*.

Pointcut	*Joinpoints identificados*
this(Punto)	Todos los *joinpoints* donde el objeto actual (*this*) es una instancia de *Punto* o de alguno de sus descendientes
target(Punto)	Todos los *joinpoints* donde el objeto destino (*target*) es una instancia de *Punto* o de alguno de sus descendientes
this(Elemento)&& ¡within(Elemento)	Todos los *joinpoints* donde el objeto actual es una instancia de alguna subclase de *Elemento*. Se excluye la propia clase *Elemento*.
call(*.*(..)) && target(Elemento)*	Todas las llamadas a los métodos de instancia de la clase *Elemento* y sus subclases. Con el siguiente *pointcut call(*Elemento+.*(..))* también se capturarían las llamadas a los métodos estáticos.
call(Punto.*(..)) && this(Figura)*	Todas las llamadas a los métodos de la clase *Punto* realizadas desde un objeto de la clase *Figura*.
set(*.*) && target(Linea)*	Todas las asignaciones sobre cualquier atributo de la clase *Linea*.

Cuadro A.8: Ejemplos de *pointcuts* basados en objetos.

Para ciertos *joinpoints* no existen los objetos *this* o *target*, o bien representan al mismo objeto. La Tabla A.9 define los objetos *this* y *target* para cada categoría de *joinpoint*, en el caso que existan.

Joinpoint	**Objeto *this***	**Objeto** *target*
Llamada a método	objeto que realiza la llamada	objeto que recibe la llamada
Ejecución de método	objeto que ejecuta el método	
Llamada a constructor	objeto que realiza la llamada	No existe
Ejecución de constructor	objeto que ejecuta el constructor	
Lectura de atributo	objeto que realiza la lectura	objeto al que pertenece el atributo
Asignación de atributo	objeto que realiza la asignación	objeto al que pertenece el atributo
Ejecución de manejador	objeto que ejecuta el manejador	
Inicialización de clase	No existe	No existe
Inicialización de objeto	objeto que realiza la inicialización	
Pre-inicialización de objeto	No existe	No existe
Ejecución de *advice*	*aspecto* que ejecuta el *advice*	

Cuadro A.9: Objetos *this* y *target* según la categoría del *joinpoint*.

Cabe remarcar que no existe objeto *this* en contextos estáticos como el cuerpo de un método o un inicializador estático. Ni tampoco existe objeto *target* para *joinpoints* asociados con métodos o campos estáticos.

A.2.1.2.5. Pointcuts basados en argumentos

Estos *pointcuts* capturan los *joinpoints* cuyos argumentos son de un cierto tipo mediante el descriptor *args*. También son utilizados para exponer el contexto de los *joinpoints*, en concreto sus argumentos. Existe un único descriptor de este tipo cuya sintaxis es la siguiente:

```
args(typeSignature o <identifier>, typeSignature o <identifier>)
```

La Tabla A.10 define los argumentos para cada categoría de puntos de enlace.

Joinpoint	**Argumentos**
Llamada a método	argumentos del método
Ejecución de método	argumentos del método
Llamada a constructor	argumentos del constructor
Ejecución de constructor	argumentos del constructor
Lectura de atributo	no existen
Asignación de atributo	nuevo valor del atributo
Ejecución de manejador	excepción manejada
Inicialización de clase	no existen
Inicialización de objeto	argumentos del constructor
Pre-inicialización de objeto	argumentos del constructor
Ejecución de *advice*	argumentos del *advice*

Cuadro A.10: Argumentos según la categoría del *joinpoint*.

A.2.1.2.6. Pointcuts basados en condiciones

Este tipo de *pointcuts* identifica los *joinpoints* que cumplen una cierta expresión condicional expresada mediante el descriptor *if*. Existe un único descriptor de *joinpoint* condicional cuya sintaxis es la siguiente:

```
if(expresionBooleana)
```

A.2.2. Advices

Los *advices* son construcciones similares a los métodos de una clase. Especifican las acciones a realizar en los *joinpoints* que satisfacen cierto *pointcut*.

Los *advices* aparecen en el cuerpo de un *aspecto* y no tienen un nombre o identificador como es el caso de los *pointcuts* o los métodos de una clase, ya que no será necesario referirnos a ellos en el código.

A.2.2.1. Sintaxis

Un *advice* consta de tres partes: declaración, especificación del *pointcut* y cuerpo. La declaración de un *advice* está delimitada por el carácter ":" consta del tipo de retorno (sólo para avisos de tipo *around*), el tipo de aviso (*before*, *after* o *around*) y su lista de parámetros, que expone información de contexto para que pueda ser usada en el cuerpo del mismo. Después de la lista de parámetros se pueden especificar las excepciones que puede lanzar el cuerpo del *advice* mediante la cláusula *throws*.

La especificación del *pointcut* puede ser anónimo, o bien el identificador de un *pointcut* definido previamente, con sus correspondientes parámetros, si los hay. Por último, el cuerpo de un *advice* encierra las acciones a realizar cuando se alcanzan los *joinpoint* especificados.

La figura A.2 muestra la estructura de un *advice* asociado a un *pointcut* con nombre. Se trata del *pointcut notificarFinTarea* correspondiente a la figura A.1 utilizado en el *aspecto Colaboración* (componente de apoyo al trabajo en grupo para J2ME -véase *Capítulo 8*-).

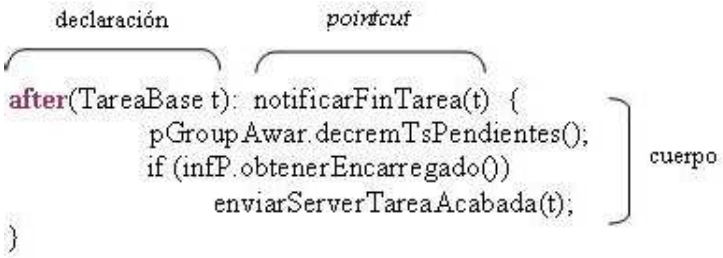


Figura A.2: Definición de un *advice* asociado a un *pointcut* con nombre.

A.2.2.2. Tipos de advices

El tipo de un *advice* matiza en qué momento se ejecutará el código asociado, en relación con el *joinpoint* capturado. *AspectJ* soporta tres tipos de *advices*: *before*, *after* y *around*

A.2.2.2.1. Advice *before*

Es el tipo de *advice* más simple. Como su nombre indica, el cuerpo del *advice* se ejecuta antes del *joinpoint* capturado. Si se produjera una excepción durante la ejecución del *advice*, el *joinpoint* capturado no se ejecutaría.

A.2.2.2.2. Advice *after*

En este caso el cuerpo del *advice* se ejecuta después del *joinpoint* capturado.

Dado que en muchas ocasiones es necesario diferenciar entre situaciones donde el control retorna normalmente y aquellas en las que se ha producido una excepción, *AspectJ* provee mecanismos para distinguirlas. Existen estas tres variantes de *advices after* para discernir los distintos casos:

- *Advice* ***after*** (no calificado): el *advice* se ejecutará siempre, sin importar si el *joinpoint* finaliza normalmente o con el lanzamiento de alguna excepción. Se comporta, por tanto, como un bloque *finally* de Java. En el ejemplo reflejado en la figura A.2 el código del cuerpo del *advice* se ejecutará tras la ejecución de todo *joinpoint* capturado por el *pointcut* *notificarFinTarea* (véase figura A.1).

- *Advice* ***after returning***: el *advice* sólo se ejecutará si el *joinpoint* termina normalmente, pudiendo acceder al valor de retorno devuelto por éste. Si el *joinpoint* termina con una excepción, el *advice* no se ejecutará.

En el siguiente ejemplo el método *Logger.writeLog* se ejecutará tras la ejecución de todo *joinpoint* capturado por el *pointcut* *cambioPosicionPunto* que termine normalmente.

```
after(Punto p) returning: cambioPosicionPunto(p){
        Logger.writeLog("Cambio posición Punto:" + p.toString());
}
```

- *Advice* ***after throwing***: el *advice* sólo se ejecutará si el *joinpoint* finaliza con el lanzamiento de una excepción, pudiendo acceder a la excepción lanzada. Si el *joinpoint* finaliza normalmente, el cuerpo del *advice* no se ejecutará.

En el siguiente ejemplo el cuerpo del *advice* se ejecutará tras las llamadas a los métodos de cualquier clase que lancen la excepción *RemoteException*.

```
after() throwing (RemoteException ex): call(* *.*(..) throws RemoteException){
        System.out.println("Exception capturada:" + ex);
}
```

A.2.2.2.3. Advice *around*

Es el tipo de *advice* más complejo, pero a la vez más potente. Se ejecutan en lugar del *joinpoint* capturado, ni antes ni después. Por lo tanto, rodea la ejecución del *joinpoint*, el cual no se ejecuta a menos que en el cuerpo del *advice* se explicite su invocación mediante la palabra clave específicamente destinada a este fin: la instrucción ***proceed***, la cual funciona como una invocación a un método. Este método toma cómo parámetros los definidos en la lista de parámetros del *advice* y devuelve un valor del tipo especificado en el mismo.

Mediante este tipo de *advice* se puede:

- remplazar la ejecución original del punto de enlace por otra alternativa
- omitir la ejecución del *joinpoint*, por ejemplo cuando los parámetros de un método son ilegales.
- alterar el contexto sobre el que se ejecuta el *joinpoint*: los argumentos, el objeto actual (*this*) o el objeto destino (*target*) de una llamada, a través del uso de *proceed*.
- envolverlo con operaciones previas y posteriores a su ejecución.

Puesto que los *advices around* sustituyen al *joinpoint* capturado, deberán devolver un valor de un tipo compatible al tipo de retorno del *joinpoint* remplazado, que puede ser *Object*.

El siguiente ejemplo de *advice around* envuelve la ejecución de los *joinpoints* capturados a través del *pointcut comunicarTareaCancelada* mostrado anteriormente. Este *pointcut* es utilizado en el *aspecto Colaboración* (componente de apoyo al trabajo en grupo para J2ME -véase *Capítulo 8*-).

```
void around(TareaBase t): comunicarTareaCancelada(t) {
        EnvioAvisosSMS(t);
        proceed();
        particGAwar.incremTsCanceladas();
        particGAwar.decremTsPendientes();
        enviarServerTareaCancelada(t);
}
```

A.3. Implementación transversal estática

Si bien los *advices* permiten modificar el comportamiento de los *joinpoints*, en ocasiones es necesario alterar la estructura estática de los programas. Esto incluye alterar las jerarquías de tipos y clases, agregar métodos y atributos a las clases del programa base. Son cambios que afectan el comportamiento en compilación del código base y dan soporte a la *implementación transversal dinámica*. Eso es posible a través de la *implementación transversal estática*.

Existen diversas construcciones de este tipo, a saber: *declaraciones inter-tipo*, *declaraciones de parentesco*, *declaraciones de precedencia*, *declaraciones en tiempo de compilación* (permiten añadir advertencias o errores en tiempo de compilación para notificar ciertas situaciones que deseamos advertir o evitar), y *excepciones suavizadas* (permiten ignorar el sistema de chequeo de excepciones de Java, silenciando las excepciones que tienen lugar en determinados *joinpoints*). No obstante, tan sólo se presentan aquí las tres primeras, al ser las que han sido empleadas en la construcción del FPI.

A.3.1. Declaraciones inter-tipo

Las *declaraciones inter-tipo*, también llamadas *introducciones*, permiten agregar comportamiento y estructura directamente en las clases de los objetos afectados. También permiten alterar la jerarquía de tipos de la aplicación base añadiendo miembros (campos, métodos o constructores) a clases, interfaces o *aspectos* de una aplicación, sin modificar su código. A continuación se introducen algunos detalles y ejemplos.

Cualquier miembro inter-tipo (campo, método o constructor) añadido a cierta clase, *aspecto* o interfaz será heredado por las subclases de la clase o *aspecto*, o en el caso de que el tipo destino sea una interfaz, por las clases que la implementen.

Introducción de campos

La declaración para añadir un campo *color* a la clase *Punto* será la siguiente:

```
public java.awt.Color Punto.color = java.awt.Color.black;
```

Introducción de métodos concretos

Se puede añadir un método concreto a cualquier tipo, incluido interfaces. Por ejemplo, para añadir los métodos *setColor* y *getColor* a la clase *Punto* utilizaremos las siguientes declaraciones:

```
public void Punto.setColor(java.awt.Color c){
        color = c;
}
public java.awt.Color Punto.getColor(){
        return color;
}
```

Así, en la construcción del FPI, el *aspecto IncrementadorEstructura* introduce los métodos *hashCode*() y *equals()*, a fin de poder trabajar con la tabla de *hash* (componentes de personalización de una pantalla para MIDP; véase *Capítulo 8*).

```
public int EntidObjet.hashCode() { ... }
```

Introducción de métodos abstractos

También se puede añadir un método abstracto a tipos abstractos (clases o *aspectos*) y a interfaces. La palabra clave *abstract* puede ir antes o después de los modificadores de acceso.

Por ejemplo, la declaración para añadir el método abstracto *toXML* a la clase abstracta *XMLSupport* será la siguiente.

```
public abstract void XMLSupport.toXML(PrintStream out);
```

Introducción de constructores

Se pueden añadir constructores tanto a clases abstractas como concretas, pero no a interfaces o *aspectos*. Con la siguiente declaración se añade un constructor a la clase *Punto*:

```
public Punto.new(int x, int y, java.awt.Color color){
        this(x,y);
        this.color = color;
}
```

A.3.2. Declaraciones de parentesco

Este mecanismo de *implementación estática* nos permite establecer que ciertas clases implementan nuevas interfaces o extienden nuevas clases.

La sintaxis es *declare parents*.

Implementar nuevas interfaces

Puesto que mediante este tipo de declaraciones se indica que una clase implementa una nueva interfaz, es necesario implementar los métodos definidos en el cuerpo de la interfaz. Para ello se puede recurrir a *declaraciones inter-tipo*.

En el siguiente ejemplo se usa una declaración de parentesco para indicar que la clase *Punto* implementa la interfaz *Cloneable* y mediante una declaración inter-tipo se añade a la clase *Punto* la implementación del método *clone*.

```
declare parents: Punto implements Cloneable;

public Object Punto.clone() throws CloneNotSupportedException {
        Punto p = (Punto) super.clone();
        return p;
}
```

A diferencia de las declaraciones inter-tipo, en las declaraciones de parentesco sí se pueden usar patrones de tipo para que éstas afecten a más de una clase destino. Por ejemplo, la siguiente declaración indica que todas las clases del paquete *com.company* implementan la interfaz *Serializable*.

```
declare parents: com.company..* implements Serializable
```

Extender nuevas clases

Con la construcción *declare parents* también se puede especificar que ciertas clases extienden nuevas clases.

Por ejemplo, mediante la siguiente declaración indicamos que la clase *Perfil* hereda de la clase *EntidObjetBase* (componente de personalización de pantalla, objetivo de personalización 'valores por defecto de un formulario' en el sistema "LectorNoticiasAumentado"; caso de estudio tratado en el *Capítulo 7*). Esta declaración aparecería en el *aspecto IncrementadorEstructura* al instanciar el FPI para dicha aplicación.

```
declare parents: Perfil extends EntidObjetBase
```

En el caso de que especifiquemos más de una clase en la lista de clases a extender, éstas deben pertenecer a la misma jerarquía de herencia. Además, debe hacerse sin violar las reglas de herencia de Java (clase no puede ser padre de una interfaz y no se pueden hacer declaraciones que den lugar a herencia múltiple).

A.3.3. Declaraciones de precedencia

En ocasiones, dos *advices* definidos en *aspectos* diferentes afectan a los mismos *joinpoints* y por lo tanto, es interesante establecer una relación de precedencia entre ellos, de manera que uno se ejecute antes que otro. Este tema ha sido tratado en el *Capítulo 6* (véase *Capítulo 6*; *sección 6.2.3.3.4*).

Mediante las declaraciones de precedencia, cuya construcción utiliza las palabras clave *declare precedence*, se puede especificar que todos los *advices* de un *aspecto* tengan preferencia a la hora de ejecutarse con respecto a los de otros *aspectos*.

El carácter "*"por sí solo en la lista de *aspectos* representa a cualquier *aspecto* no listado, indicando que la precedencia de los *aspectos* no listados es antes, entre o después de los *aspectos* que aparecen en la lista, en términos de precedencia.

Por ejemplo, en la siguiente declaración el *aspecto Aspecto* es el de mayor precedencia, *LogAspect* el de menor, y con una precedencia intermedia se encuentran el resto de *aspectos* no listados.

```
declare precedence: Aspecto, *, LogAspect;
```

Si queremos dar mayor precedencia a los *aspectos* que extienden cierto *aspecto* se puede utilizar el comodín "+", tal y como se hace en la siguiente instrucción:

```
declare precedence: TraceAspect+, *;
```

A continuación se presenta un ejemplo sencillo de programa "Hola mundo" en *AspectJ*. Este ejemplo sirve también para mostrar la estructura de una construcción *aspecto*, en la que se encapsulan todas las construcciones requeridas, siguiendo el patrón de declaración de clases (la palabra clave *class* es sustituida por la palabra clave *aspect*). En este caso la

unidad de programa *aspecto*, de nombre *Aspecto*, tan sólo incluye dos *advices* asociados a dos *pointcuts* anónimos, los cuales interceptan un mismo método (el método *main*).

La definición de la clase se incluye en un programa fuente de nombre *Clase.java*, y la definición del aspecto en otro distinto denominado *Aspecto.aj*.

```
/**
* Clase.java
*/
public class Clase {
        public static void main(String args[]) {
                System.out.println("Hola mundo");
        }
}
/**
* Aspecto.aj
*/
public aspect Aspecto {
        before():execution( void Clase.main(..) ) {
                System.out.println("Inicio metodo main");
        }
        after():execution( void Clase.main(..) ){
                System.out.println("Fin metodo main");
        }
}
```

A.4. Acerca de los mecanismos de herencia

Un *aspecto* se puede extender de diferentes formas:

- construyendo un *aspecto* abstracto (modificador *abstract*) que sirve de base para otros aspectos. Un *aspecto* no puede heredar de *aspectos* concretos, tan sólo se puede heredar *aspectos* abstractos.

Cuando un aspecto hereda un aspecto abstracto, no sólo hereda los atributos y métodos del aspecto sino también los *pointcuts* y los *advices*.

- heredando de clases o implementando una serie de interfaces. Sin embargo, *AspectJ* no permite a una clase heredar de un *aspecto*.

Apéndice B

Esquemas de solución genéricos para la construcción del *Framework de Plasticidad Implícita*

En este apéndice se proporcionan los detalles de diseño e implementación correspondientes a las cinco componentes desarrolladas en los casos de estudio (*Capítulo 7*), expresados de manera genérica, esto es, sustituyendo las referencias específicas del sistema por referencias genéricas. Así, para hacer referencia de forma genérica a la clase que representa la *entidad objetivo* involucrada en cada sistema se utiliza el identificador de clase *EntdObj*. Por otro lado, todas las particularidades identificadas en la descripción de los casos de estudio han sido generalizadas con el propósito de obtener un diseño y una implementación suficientemente genéricos para su validez ante diversas situaciones y necesidades.

Las cinco componentes descritas son: personalización de una pantalla, la cual se describe para los dos objetivos trabajados en los casos de estudio, personalización de una funcionalidad del sistema base, la componente de apoyo al trabajo en grupo y la componente de adaptación al entorno o recurso hardware.

B.1. Componentes genéricas de personalización de una pantalla

En esta sección se describe una generalización del código aportado para dar solución a los objetivos de personalización de una pantalla sobre aplicaciones desarrolladas en J2ME, que es el tipo de aplicaciones contempladas en los casos de estudio presentados en el *Capítulo 7*. Sin embargo, de cara a proporcionar código reutilizable en otras plataformas, el framework contempla la distinción entre aplicaciones móviles y fijas, proporcionando una versión para cada caso (véase *sección 7.2.2.1.*).

Al igual que se hace en el *Capítulo 7*, se utilizan los nombres *PantallaLista*, *PantallaFormulario* y *FormularioAsistidoporLista* para hacer referencia a las pantallas involucradas.

B.1.1. Objetivo de personalización 'Valores por defecto en un formulario'

El propósito de este objetivo de personalización se describe en detalle en el *Capítulo 7* (véase *Capítulo 7*; *sección 7.1.3.1.1.*).

B.1.1.1. Capa sensible al contexto

Las clases que se proponen en la versión genérica para capa son las siguientes:

Clase *UsoEntdObj*

- *Objetivo*: soportar en memoria cada posible instancia de la clase *EntdObj*, juntamente con el número de ocurrencias registradas de la misma y la fecha en que se produjo por última vez.
- *Atributos*:
  - *EntdObj*: objeto de la clase que representa la *entidad objetivo* del sistema subyacente.
  - *contadorUsos*: el contador del número de ocurrencias de cada instancia de la clase *EntdObj*.
  - *ultimaOcurrencia*: fecha -clase *Date* en java- correspondiente a la última ocurrencia producida de la *entidad objetivo* en cuestión.
- *Métodos*:

- *Constructora con parámetros*: se recibe como parámetro la *EntdObj* a ser inicializada y el valor inicial para el contador. En el campo fecha se asigna la fecha actual.
- *incOcurrencias*: se incrementa en uno el contador y se actualiza la fecha a la actual.
- *obtenerContador*: retorna el valor del contador.
- *obtenerUltimoUso*: retorna el valor de la fecha correspondiente al último uso.
- *obtenerClave*: retorna el objeto *EntdObj* de que se compone.
- *serializar*: convierte los valores de una instancia de la clase en datos a bajo nivel para ser almacenados de manera persistente.
- *materializar*: operación inversa a la anterior. Convierte los datos almacenados de manera persistente en valores de programa adecuados para conformar una instancia de la clase.

Clase *TablaUsosEntdObjconModa*

- *Objetivo*: registrar, contabilizar y soportar en memoria las ocurrencias de cada combinación de parámetros y el valor de moda en curso. Es también la responsable de pasar la información registrada de manera persistente a la memoria volátil al inicio del programa, así como de volver a registrarla de forma persistente al finalizar la ejecución.
- *Atributos*:
  - *tablaUsos*: se trata de la tabla de *hash* propiamente dicha.
  - *EntdObjModa*: contiene el valor correspondiente al objeto de la clase *EntdObj* de moda, esto es, aquella instancia que se ha repetido un mayor número de ocasiones (instancia 'de moda').
  - *contadorModa*: valor del contador de ocurrencias para la instancia 'de moda' de la clase.
  - *cambios*: valor lógico que indica si durante la sesión se han producido o no cambios en el registro de ocurrencias.
- *Métodos*:
  - *Constructora sin parámetros.*
  - *ObtenerEntdObjModa*: retorna el valor del atributo *entdObjModa*.

- *ObtenerContadorModa*: retorna el valor del atributo *contadorModa*.
- *cargarTabla*: transfiere el registro de ocurrencias del que se dispone en memoria persistente a la tabla de *hash*, materializando todos los registros en objetos de la clase *UsoEntdObj*. Además, conforme se va construyendo la tabla se va calculando la entidad objetivo de moda - máximo valor en el contador-, a guardar en el atributo *entdObjModa*. Esta operación debe realizarse al inicio de la ejecución.
- *almacenarTabla*: transfiere el registro de ocurrencias del que se dispone en la memoria volátil a la memoria persistente, esto es, serializa todos los objetos de la clase *UsoEntdObj*. Esta operación debe realizarse al finalizar la ejecución. En ocasiones, y dependiendo de la amplitud de muestreo considerada en cada caso, se aplica un filtro para registrar únicamente las ocurrencias que no hayan caducado, esto es, que no haya transcurrido el periodo de muestreo desde su última ocurrencia.
- *actualizarOcurrencia*: comprueba si ya se había producido anteriormente alguna ocurrencia de esa *EntdObj*. Si es el caso se incrementa el contador. En caso contrario crea una entrada para la misma. Además, comprueba si la entidad objetivo de moda –atributo *EntdObjModa*- o el contador asociado –*contadorModa*- requieren ser actualizados como consecuencia de la nueva ocurrencia producida.

En el diagrama de clases 8.6 del *Capítulo 8* (véase *Capítulo 8*; *sección 8.2*), correspondiente a la versión proporcionada por el FPI, quedan reflejados todos estos detalles.

B.1.1.2. Capa aspectual

Los *aspectos* que se proponen en la versión genérica para personalizar los valores por defecto de un formulario (denominado genéricamente *PantallaFormulario*), de acuerdo a la información representada en el *modelo contextual*, son el *aspecto IncrementadordeOperaciones* y el *aspecto CapturadorAdaptador*, que a continuación se presenta en detalle..

En lo que respecta al *aspecto IncrementadordeOperaciones*, debe incorporar, al menos, los métodos *equals* y *hashCode* en las clases genéricas introducidas para las *entidades objetivo*, a fin de poder trabajar con la tabla de *hash*. En ocasiones estos métodos deberán ser especializados en las propias clases efectivas. Esto se llevaría a cabo a través de este mismo aspecto. Más detalles acerca de este *aspecto* se proporcionan en el *Capítulo 7*.

Aspecto *CapturadorAdaptador*

- *Objetivo*: capturar las combinaciones de valores introducidos por el usuario en el formulario *PantallaFormulario*, con el propósito de personalizar los valores por defecto a mostrar en la misma de acuerdo a la heurística aplicada.
- *Atributos*: un objeto de la clase *TablaUsosEntdObjconModa*, en la que registrar y mantener las combinaciones de valores capturados, juntamente con la gestión de sus ocurrencias.
- *Puntos de corte*:
  - *capturarCombinación*, el cual tiene una función de captura.
    - *puntos de unión*: se intercepta la ejecución del método encargado de procesar el propio formulario una vez introducido, y se captura el objeto propietario del método, esto es, la *PantallaFormulario* a través de *this* (véase *Apéndice A*).
    - *consejo asociado*: antes de proceder al procesamiento del formulario se obtiene la combinación de valores introducida por el usuario –invocación a un método de la aplicación base, que de no formar parte del sistema es incorporado a través del *aspecto IncrementadordeOperaciones* (método *obtenerEntidObjet* en el diagrama de clases de las figuras 8.8 y 8.9 (véase *Capítulo 8*)- con objeto de registrar esa ocurrencia en la tabla de *hash* –invocación al método *actualizarOcurrencia* de la clase *TablaUsosEntdObjconModa*.
  - *establecerValoresporDefecto*, el cual tiene una función de adaptación.
    - *puntos de unión*: se intercepta la ejecución del método constructor de la pantalla *PantallaFormulario*, y se captura el objeto propietario del método, esto es, la *PantallaFormulario* a través de *this* (véase *Apéndice A*).
    - *consejo asociado*: una vez finalizada la ejecución del método constructor interceptado se obtiene el valor de la entidad objetivo de moda en ese momento –invocación al método *ObtenerEntdObjModa* de la clase *TablaUsosEntdObjconModa*-, a fin de cargar esa combinación como valores por defecto a mostrar como consecuencia del proceso de personalización. El método encargado de cargar unos valores determinados como combinación de valores por defecto en la pantalla *PantallaFormulario*, puede que no forme parte de la aplicación original. En ese caso es incorporado por el *aspecto IncrementadordeOperaciones* (método establecerEntObjPorDefecto en el diagrama de clases de las figuras 8.8 y 8.9- véase *Capítulo 8*-).
  - *materializarTabla*, cuya responsabilidad consiste en cargar en memoria la relación de todas las ocurrencias registradas y almacenadas de manera per-

sistente, información a soportar a través de la tabla de *hash* representada por la clase *TablaUsosEntdObjconModa*. Este *punto de corte* se activa al inicio del programa.

- *puntos de unión*: se intercepta la ejecución de uno de los métodos iniciales de la ejecución del sistema.
- *consejo asociado*: carga la tabla en memoria –invocación al método *cargarTabla* de la clase *TablaUsosEntdObjconModa*.

- *serializarTabla*, encargado de almacenar de forma persistente las ocurrencias recogidas en la tabla de *hash* antes de finalizar el programa.
  - *puntos de unión*: se captura la ejecución de uno de los métodos finales de la ejecución del sistema.
  - *consejo asociado*: almacena la tabla de manera persistente –invocación al método *almacenarTabla* de la clase *TablaUsosEntdObjconModa*.

En los diagramas de clase 8.9 y 9.10 del *Capítulo 8* (véase *Capítulo 8*; *sección 8.3*), correspondientes a las versiones proporcionadas por el FPI, quedan reflejados todos estos detalles.

B.1.2. Objetivo de personalización 'Ordenación de una lista'

El propósito de este objetivo de personalización se describe en detalle en el *Capítulo 7* (véase *Capítulo 7*; *sección 7.1.3.1.3.* y *7.2.3.1.3.*).

B.1.2.1. Capa sensible al contexto

Las clases que se proponen para mantener tanto en la memoria volátil como en la memoria persistente el registro del número de ocasiones en que el usuario opta por cada opción de la lista, a fin de poder aplicar una cierta heurística, son las siguientes:

Clase *UsoEntdObjComparable*

La única diferencia de esta clase con respecto a la *UsoEntdObj* del objetivo de personalización *'valores por defecto en un formulario'* es que en este caso implementa una interfaz para comparar el objeto propietario con otro objeto de la misma clase recibido como parámetro, retornando un entero, a efectos de proceder a la ordenación de todos los objetos de la tabla de *hash*. Se trata de la interfaz *Comparable*.

A continuación se menciona únicamente lo que aporta esta clase respecto a la *UsoEntdObj* anterior.

- *Objetivo*: soportar en memoria cada posible instancia de la *EntdObj*, juntamente con el número de ocurrencias registradas de la misma y la fecha en que se produjo por última vez.

  Está especialmente preparada para comparar dos objetos de la clase entre sí, a fin de proceder a su ordenación utilizando la clase *QuickSort* (implementación del método *comparar*).

- *Atributos*: Exactamente los mismos que en la clase *UsoEntdObj* anteriormente descrita.

- *Métodos*: Los mismos que en la clase *UsoEntdObj*, además del método siguiente para implementar la interfaz *Comparable*:

  - *comparar*: utilizado en la clase *QuickSort* para comparar los distintos elementos del vector con el pivote en la aplicación del algoritmo de ordenación *quicksort*. El valor de referencia en la ordenación es el número de ocurrencias, uno de los atributos de la clase.

Clase *TablaUsosEntdObjComparable*

A pesar de que esta clase es en esencia equivalente a la clase del objetivo de personalización *'valores por defecto en un formulario'* –la clase *TablaUsosEntdObjconModa*-, no obstante, al tratarse de otro objetivo de personalización plantea ciertas diferencias. Una de ellas es que en su representación interna no mantiene el valor de la *entidad objetivo* de moda. Eso implica eliminar parte de los métodos, así como también la gestión pertinente para mantener la *entidad objetivo de moda* actualizada a través de los métodos *materializarTabla* y *serializarTabla*.

Otra diferencia es que, dado que este objetivo de personalización requiere ordenar las ocurrencias de las distintas entidades objetivo aplicando el algoritmo de ordenación *quicksort*, se ofrece un método específico encargado de pasar los componentes de la tabla de *hash* a un vector. A continuación se presenta en detalle.

- *Objetivo*: registrar, contabilizar y soportar en memoria el número de ocasiones en que el usuario selecciona cada una de las opciones de la lista objeto de personalización, utilizando una tabla de *hash*. Es también la responsable de pasar la información registrada de manera persistente a la memoria volátil al inicio del programa, así como de volver a registrarla permanentemente al finalizar la ejecución.

- *Atributos*: contiene dos de los cuatro atributos de la clase *TablaUsosEntdObjconModa* con exactamente el mismo propósito que el propuesto para la misma. Se trata de los atributos: *tablaUsos* y *cambios*.

- *Métodos*:

  - *Constructora sin parámetros.*

  - *cargarTabla*: operación que tiene exactamente la misma responsabilidad que el método con el mismo nombre de la clase *TablaUsosEntdObjconModa*, pero sin la parte de calcular la *entidad objetivo* de moda.

    A diferencia del objetivo de personalización 'valores por defecto en un formulario', aquí conviene realizar un pequeño paso más que consiste en acabar de completar la tabla de *hash* con los objetos de la *entidad objetivo* que, aunque no hayan sido seleccionados en ninguna ocasión por el usuario, igualmente deben aparecer en la lista *PantallaLista*. Por ello se requiere recorrer el archivo correspondiente, a efectos de añadir en la tabla de *hash* los que aún no estén incluidos, por supuesto con un valor en el contador de ocurrencias de cero. Para los dos casos de estudio trabajados estos archivos han sido la agenda de *Contactos* del móvil y el archivo de material.

    Esta consideración evita redundancias, además de prevenir que el archivo de registros correspondiente aumente desmesuradamente.

- *almacenarTabla*: exactamente la misma responsabilidad que el método con el mismo nombre de la clase *TablaUsosEntdObjconModa*. No obstante, por el mismo motivo comentado para el método anterior, en este caso tan sólo se registran en la memoria persistente aquellas entradas de la tabla cuyo contador de ocurrencias sea mayor que cero.

- *actualizarOcurrencia*: exactamente la misma responsabilidad que el método con el mismo nombre de la clase *TablaUsosEntdObjconModa*, aunque sin la parte de actualizar la *entidad objetivo* de moda.

- *obtenerVectorOrdenado*: transfiere el contenido de la tabla de *hash* a un vector, con el propósito de aplicar el algoritmo de ordenación del *quicksort*.

Clase *QuickSortDecreciente*

- *Objetivo*: ofrecer un método estático para aplicar una ordenación decreciente sobre un vector de objetos, los cuales deben cumplir la interfaz *Comparable*. El algoritmo de ordenación que se utiliza es el del *quicksort*.

- *Métodos*: ofrece dos métodos estáticos para la implementación del algoritmo de ordenación *quicksort*, denominados *quicksort* -método recursivo- y *partir*. El método de ordenación retorna tanto el vector ordenado resultante como el número de componentes totales del mismo.

Se requiere implementar estas dos últimas piezas de código cuando la componente va a ser destinada a una aplicación móvil, dado que la modalidad de Java J2ME, a diferencia de la J2SE, no ofrece algoritmos genéricos de ordenación en su biblioteca estándar. Esta es una de las limitaciones de J2ME.

Interfaz Comparable

Interfaz que declara un método que realiza la comparación entre dos objetos de la clase *UsoEntdObjComparable* (en este caso el valor a comparar es el del atributo *contadorUsos*), retornando un valor entero, útil para la aplicación del algoritmo de ordenación del *quicksort*. Se trata del método *comparar*.

En los diagramas de clase 8.7 y 8.8 del *Capítulo 8* (véase *Capítulo 8*; *sección 8.3*), correspondientes a las versiones proporcionadas por el FPI, quedan reflejados todos estos detalles.

B.1.2.2. Capa aspectual

Los *aspectos* que se proponen para ordenar de manera personalizada la lista a mostrar en la que pantalla referenciada como *PantallaLista*, según el estado del *modelo contextual*, son el *aspecto IncrementadordeOperaciones* y el *aspecto CapturadorAdaptador*, que a continuación se presenta en detalle.

En lo que respecta al *aspecto IncrementadordeOperaciones*, al igual que en el objetivo de personalización anterior, debe incorporar, al menos, los métodos *equals* y *hashCode* en las clases genéricas introducidas para las *entidades objetivo*, a fin de poder trabajar con la tabla de *hash*. Más detalles acerca de este *aspecto* se proporcionan en el *Capítulo 7*.

Aspecto *CapturadorAdaptador*

- *Objetivo*: capturar los elementos de la lista seleccionados por el usuario cada vez que accede a la pantalla *PantallaLista*, con el propósito de personalizar el orden de aparición de las distintas opciones.
- *Atributos*:

- *tabla*: un objeto de la clase *TablaUsosEntdObjComparable*, en la que depositar y mantener las ocurrencias de las opciones seleccionadas por el usuario en la lista *PantallaLista*.
- *vectorUsos*: vector al que transferir la información depositada en *tabla* para proceder a su ordenación.
- *formAnt*. Una referencia al formulario que da acceso a la lista objeto de personalización. Este atributo es necesario en la versión para móviles, a fin de mantener una referencia a la pantalla previa, de cara a resolver la navegación entre pantallas.

▪ *Puntos de corte*:

- *capturarSelección*, el cual tiene una función de captura.
  - *puntos de unión*: se intercepta la invocación –*punto de unión call*- al método que obtiene el campo del formulario en el que estamos interesados de la pantalla *FormularioAsistidoporLista*, y de ese modo conocer el valor seleccionado por el usuario.
  - *consejo asociado*: registra el valor del campo del formulario en el que se está interesado -valor retornado por el método interceptado- en la tabla de *hash* mediante la invocación al método *actualizarOcurrencia* de la clase *TablaUsosEntdObjComparable*.
    En determinados casos la pantalla *FormularioAsistidoporLista* se compone de varios campos cuya introducción puede ser facilitada invocando a la lista (*PantallaLista*) en cada uno de ellos. Es el caso del formulario *FormularioInforme* del caso de estudio *'Controlador de Obras'* (véase *Capítulo 7*; *sección 7.1.3.1.3.*), donde para cumplimentar el formulario *Informe Tarea/Jornada* es necesario introducir uno o varios materiales.
    En estos casos debe haber un punto de corte *capturarSelección* específico para cada uno de esos campos del formulario.
- *establecerOrdenOpciones*, el cual tiene una función de adaptación.
  - *puntos de unión*: se intercepta la ejecución del método constructor de la pantalla *PantallaLista*, y se captura el objeto propietario del método a través de *this* (véase *Apéndice A*).
  - *consejo asociado*: se trata de un *consejo around* (véase *Apéndice A*), dado que se requiere suplantar el método constructor con un código idéntico al ya existente en la aplicación base, excepto en la parte donde se procede a la construcción de la lista. En lugar de construirla a partir del archivo de *entidades objetivo* (archivo de material o agenda de *Contactos* en los casos de

estudio trabajados), se construye a partir del contenido de la tabla de *hash* previamente ordenado (invocación al método *obtenerVectorOrdenado*).

- *tratarOpciónPersonalizada*, el cual tiene una función de adaptación.
  - *puntos de unión*: se intercepta la ejecución del método *commandAction* –versión para aplicaciones móviles-, que es el método que se requiere implementar para cumplir la interfaz *CommandListener*, el manejador de eventos de pantalla. Además se capturan los argumentos de dicho método (un comando y una pantalla), como parte del contexto, a través de *args* (véase *Apéndice A*), así como el objeto propietario del método a través de *this*.
  - *consejo asociado*: se trata también de un *consejo around*, dado que se requiere de nuevo suplantar el método interceptado con un código idéntico al ya existente en la aplicación base, excepto en la parte donde se incluye el tratamiento a realizar cuando el usuario selecciona una opción de la lista en *PantallaLista*. En este caso la lista ha estado personalizada por parte del *aspecto CapturadorAdaptador*. En consecuencia, el tratamiento a llevar a cabo en cada opción es también competencia del *aspecto*. De hecho, este tratamiento consiste en obtener la instancia de la *entidad objetivo* que representa la opción de la lista seleccionada, para que de ello tenga constancia la pantalla *FormularioAsistidoporLista*. El resto de acciones son idénticas a las del código base.
    En cuanto al resto de ramas condicionales del método *commandAction* se puede recurrir a la instrucción *proceed*[1], propia de un *consejo around*, puesto que se puede devolver el control al programa base.
- *materializarTabla*, cuya responsabilidad es cargar en memoria la relación de todas las ocurrencias registradas y almacenadas de manera persistente, información a depositar en la tabla de *hash* representada por la clase *TablaUsosEntdObjComparable*. Este *punto de corte* se activa al inicio del programa.
  - *puntos de unión*: se captura la ejecución de uno de los métodos iniciales de la ejecución del sistema.
  - *consejo asociado*: invocación al método *cargarTabla* de la clase *TablaUsosEntdObjComparable*.
- *serializarTabla*, encargado de almacenar de forma persistente las ocurrencias recogidas en la tabla de *hash* antes de finalizar el programa.

[1]Instrucción del lenguaje *AspectJ* que devuelve el control de la ejecución al *punto de unión* de la aplicación subyacente interceptado por el *aspecto*, causando la ejecución de la operación capturada. Se utiliza en los *consejos* de tipo *around*.

- *puntos de unión*: se captura la ejecución de uno de los métodos finales de la ejecución del sistema.
- *consejo asociado*: invocación al método *almacenarTabla* de la clase *TablaUsosEntdObjComparable*.

En los diagramas de clase 8.11 y 8.12 de la *sección 8.3*, correspondientes a las versiones proporcionadas por el FPI, quedan reflejados todos estos detalles.

B.2. Componente genérica de personalización de una funcionalidad de la aplicación base

El propósito de este objetivo de personalización se describe en detalle en el *Capítulo 7* (véase *Capítulo 7*; *sección 7.2.3.1.5.*).

B.2.1. Capa sensible al contexto

En la *capa sensible al contexto* se propone una clase controladora encargada de mantener en la memoria volátil los valores implicados en la personalización del parámetro objeto de tratamiento. Este parámetro es el que regula la ejecución de una cierta funcionalidad del sistema base. Se trata de la clase *GestiónParmFuncionalidadExistente*.

La *capa aspectual* captura los distintos valores para dicho parámetro, obtenidos a partir de la observación del patrón de actuación del usuario. Se trata de determinar cuál de ellos cumple mejor un cierto criterio. Debido a que la manera como aplicar dicho criterio y de seleccionar el mejor valor candidato para el parámetro en cuestión depende de cada sistema, lo más conveniente es definir una interfaz para ofrecer los métodos que llevan a cabo las operaciones asociadas. Se trata de la interfaz *comparadorParam*. De ese modo, la signatura (encabezamiento) de todos los métodos de esta clase es conocida de antemano, lo que permite que la invocación a los mismos desde la unidad *aspecto* no esté sujeta a variaciones, proporcionando de ese modo la genericidad perseguida para la *capa aspectual*.

Otros métodos de la clase *GestiónParmFuncionalidadExistente* no varían, y por lo tanto se definen en la propia clase. En definitiva, la parte de código que contiene las variaciones propias de cada sistema está perfectamente identificada, de manera que la adaptación de esta capa a distintas aplicaciones se limita a proporcionar una implementación para los métodos de la interfaz *comparadorParam*.

Clase *GestiónParmFuncionalidadExistente*

Implementa la interfaz *comparadorParam*, y por tanto es la encargada de codificar los métodos declarados en la misma (*nuevoValorDuranteMuestreo* y *actualizarParámetros*, presentados a continuación), lo cuales contienen las variaciones propias de cada sistema. Además, incorpora un conjunto de métodos cuyo código es independiente del sistema y de la funcionalidad a personalizar (las pequeñas variaciones pueden delegarse en el uso de constantes o parámetros). Por lo tanto, esta clase se deja sin completar. A continuación se presentan algunos detalles.

- *Objetivo*: soportar en memoria los valores implicados en la personalización del parámetro asociado a una cierta funcionalidad ya existente en el sistema base.
- *Atributos*:
  - *fechaUltimoMuestreo*: objeto de la clase *Date* que indica el momento en que inició el muestreo actual, cuya duración está fijada de antemano.
  - *parametroAPersonalizar*: valor del parámetro asociado a la funcionalidad que está siendo observada. Se trata del parámetro objeto de personalización.
  - *valorDuranteMuestreo*: atributo donde se van registrando, conforme van siendo observados, los posibles candidatos a suplantar el valor del atributo *parametroAPersonalizar* al finalizar el periodo de muestreo. El valor que queda es el que optimiza el criterio aplicado.
  - *archivo*: referencia al archivo donde almacenar de sesión en sesión todos estos valores.
  - *actualizarValores*: un valor lógico que indica si se han producido cambios en alguno de los valores implicados.

Los parámetros *parametroAPersonalizar* y *valorDuranteMuestreo* se declaran como *long*. Cabe tener presente que podrían ser susceptibles de variación, por ejemplo para tratar un parámetro real. No obstante, para este tipo de parámetros suelen utilizarse valores numéricos discretos. Se ha elegido el tipo *long* para abarcar el mayor rango posible para un valor entero.

- *Métodos*:
  - *método constructor*: la primera vez que se ejecuta el sistema no existe ningún archivo con los valores implicados, y por lo tanto deben ser inicializados. A partir de la siguiente ejecución su misión es materializar los valores ya almacenados en un objeto de la clase.

- *obtenerParametroAPersonalizar*: retorna el valor del atributo *parametroAPersonalizar*.
- *registrarParametros*: se trata de sustituir el archivo anterior por uno nuevo, si es que los parámetros han variado (valor cierto en el atributo *actualizarValores*).
- *serializar*: convierte los valores implicados en un formato adecuado para ser registrados en la memoria persistente.
- *materializar*: traduce el formato utilizado para el registro persistente de los valores implicados en los tipos de los atributos de la clase.

Interfaz *comparadorParam*

Interfaz que ofrece los métodos que manejan las particularidades propias de trabajar con distintos parámetros y distintos criterios en la selección del mismo, en función del sistema y de la funcionalidad que se desea personalizar.

- *Métodos*:
  - *nuevoValorDuranteMuestreo*: si el valor proporcionado como parámetro cumple el criterio aplicado mejor que el valor actual (atributo *valorDuranteMuestreo* de la clase *GestiónParmFuncionalidadExistente*), se sustituye éste por el nuevo.
  - *actualizarParámetros*: consiste en aplicar un test inmediatamente a continuación de la ejecución de la funcionalidad objeto de personalización, consistente en comprobar si ha expirado o no el periodo de muestreo. En caso afirmativo se suplanta el valor del atributo *parametroAPersonalizar* de la clase *GestiónParmFuncionalidadExistente* por el de *valorDuranteMuestreo*. Adicionalmente inicializaría de nuevo los valores implicados, con objeto de iniciar un nuevo muestreo.

En el diagrama de clases 8.13 del *Capítulo 8* (véase *Capítulo 8*; *sección 8.3*), correspondiente a la versión proporcionada por el FPI, quedan reflejados todos estos detalles.

B.2.2. Capa aspectual

Las unidades de programa *aspecto* que se han propuesto para personalizar el parámetro asociado a una cierta funcionalidad del sistema base son el *aspecto IncrementadordeOperaciones* –su presencia está condicionada a la necesidad de incorporar o no nuevos métodos, dependiendo de cada sistema- y el *aspecto CapturadorAdaptador*, que a continuación se presenta en detalle.

Aspecto *CapturadorAdaptador*

- *Objetivo*: capturar el patrón de actuación del usuario asociado a la funcionalidad objeto de personalización, con el propósito de ajustar el parámetro asociado a la realización de dicha funcionalidad.
- *Atributos*:
  - *gestor*: un objeto de la clase *GestiónParmFuncionalidadExistente*, en la que depositar y mantener los valores implicados con la funcionalidad a personalizar. El propio método constructor de esta clase, invocado en el *aspecto*, se encarga de materializar en memoria los datos relacionados con la gestión de este parámetro, almacenados de manera persistente a efectos de garantizar su mantenimiento a lo largo de todo el periodo de muestreo. De ese modo se garantiza la carga de los datos en memoria al inicio del programa.
- *Puntos de corte*:
  - *ejecutarFuncionalidad*, el cual tiene una función de adaptación.
    - *puntos de unión*: se intercepta la ejecución del método que lleva a cabo la funcionalidad en la que se está interesado de la aplicación base.
    - *consejo asociado*: una vez finalizada la ejecución de la misma procede a actualizar los parámetros implicados y a registrarlos persistentemente (métodos *actualizarParámetros* y *registrarParametros*, respectivamente, de la clase *GestiónParmFuncionalidadExistente*).
  - *proporcionarNuevoParametro*, el cual tiene una función de adaptación.
    - *puntos de unión*: se intercepta la ejecución del método que retorna el valor establecido para el parámetro a personalizar. Se trata de retornar el valor manejado por esta componente, en lugar del valor preestablecido por la aplicación. Se considera la existencia de una clase encargada de encapsular los valores de los parámetros, a la que se le llama comúnmente *Configuración*.
    - *consejo asociado*: se sustituye el valor de retorno del método interceptado, valor que corresponde al parámetro a personalizar, por el nuevo valor, esto es, por el atributo *parametroAPersonalizar* de la clase *GestiónParmFuncionalidadExistente* (invocación al método *obtenerParametroAPersonalizar*).
  - *capturarComportam*, el cual tiene una función de captura.
    - *puntos de unión*: se intercepta la invocación al método que maneja el comportamiento del usuario en relación a la funcionalidad objeto de tratamien-

to. En el caso de estudio trabajado consiste en interceptar el método encargado de cambiar el estado de una noticia de no leída a leída. Se captura también el contexto a través de *target* (véase *Apéndice A*).

- *consejo asociado*: se trata de comprobar si el nuevo valor capturado cumple el criterio requerido mejor que el valor obtenido hasta entonces a lo largo de este muestreo (invocación al método *nuevoValorDuranteMuestreo* de la clase *GestiónParmFuncionalidadExistente*), caso en el que sería convenientemente actualizado.

En los diagramas de clase 8.14 y 8.15 del *Capítulo 8* (véase *Capítulo 8*; *sección 8.2*), correspondientes a las versiones proporcionadas por el FPI, quedan reflejados todos estos detalles.

B.3. Componente genérica de apoyo al trabajo en grupo

El propósito de esta componente se describe en detalle en el *Capítulo 7* (véase *Capítulo 7*; *sección 7.1.3.2.1.*), donde se expone la misión para la que ha sido diseñada a través de la descripción de diversos escenarios.

B.3.1. Capa sensible al contexto

En esta capa interviene como mínimo la clase que representa la *consciencia de grupo particular*, a la que se le denomina *cGrupoParticular* en la versión genérica. Adicionalmente, y dependiendo de los casos, la información a manejar en esta componente debe estar complementada con otro tipo de información extra, a representar a través de otras clases adicionales. A continuación se presentan tanto la clase *cGrupoParticular* como las clases extra utilizadas en el caso de estudio del *Capítulo 7* mencionado.

Clase *cGrupoParticular*

- *Objetivo*: servir de registro y consecuente control de todas las acciones llevadas a cabo en beneficio del grupo, así como del estado o progreso de la propia actividad, desde una perspectiva individual.
- *Atributos*: algunos de ellos de uso general; otros propios de cada sistema particular.
  - *estado*: descriptor de la situación en la que se encuentra el miembro del grupo de trabajo en relación al desarrollo de la actividad que le ha sido asignada. En general, pueden considerarse tres posibles estados: activo, en espera y ocioso.

- *numTareasPendientes*: un valor entero con la función de contador. Es habitual que el trabajo de grupo a desempeñar pueda ser desglosado y contabilizado a través de la entidad *tarea*. Es por ello que, con el fin de controlar el trabajo pendiente de realizar por cada miembro del grupo, se considera adecuado contabilizar el número de tareas pendientes, como índice de medida genérico.
- *TareasCanceladas*: un valor entero con la función de contador. Aunque este atributo puede parecer específico del sistema, siguiendo con el razonamiento anterior, en muchas ocasiones se va a trabajar con una unidad de cuantificación universal que es la *tarea*. Bajo esta premisa, es adecuado considerar también la posibilidad de aplicar cancelaciones sobre las mismas.
- *notifTareasCanceladas*: un valor entero con la función de contador. Este atributo responde al mismo razonamiento anterior. La diferencia entre ambos es que el anterior contabiliza las tareas que el propio individuo ha cancelado, mientras que éste contabiliza las tareas que han sido canceladas por sus compañeros, de las cuales ha recibido una notificación.
- *avisosStock*: un valor entero con la función de contador, en este caso del número de avisos de reposición del stock en algún material por parte del usuario. Este atributo será específico para sistemas en los que se manejen y consuman materiales.

- *Métodos*:

  - *método constructor*: comporta la inicialización de todos los contadores y atributos relativos al estado del usuario. Además, cabe recordar que, tal y como se ha mencionado, esta clase se define como un *Singleton*, lo que significa que viene provista del método de obtención de su única instancia, el método *obtenerInstancia*.
  - *cambiarEstado*: modifica el descriptor de la situación en la que se encuentra el usuario respecto a la actividad grupal.
  - *decrementarTareasPend*: decrementa el contador pertinente cada vez que finaliza una tarea.
  - *obtenerTareasPend*: retorna el valor del atributo *numTareasPendientes*.
  - *incrTareasCanceladas, incrNotific* e *incrAvisosStock*: los tres métodos encargados de ir variando los tres contadores asociados conforme se van alcanzando cada una de las situaciones.
  - *serializarXML*: método encargado de serializar la información recogida en la instancia de la clase, a fin de ser enviada al servidor a través de una petición.

El resto de clases a intervenir en esta capa, si se requieren, suelen ser específicas de la aplicación. No obstante, se prevé bastante común el proveer de la información extra que se describe a través de las siguientes clases.

Clase *EquipoTarea*

Esta es una de las clases incorporadas específicamente para soportar la información adicional requerida para llevar a cabo las acciones extra de apoyo al trabajo en grupo.

- *Objetivo*: ofrecer la información relativa a los miembros del grupo que integran cada equipo de trabajo en la realización de cada una de las tareas. Esta información se obtiene de una fuente de almacenamiento persistente facilitada expresamente por el departamento de administración.
- *Atributos*:
  - *código*: es el código de una tarea, coincidente con la designación de código de tarea utilizado en cualquier otra fuente de información relativa a las tareas.
  - *numColegas*: un valor entero para conocer el número de componentes de un equipo de trabajo.
  - *ContactosEquipo*: vector que contiene los números de teléfono de todos los miembros del equipo de trabajo. Esta información resulta útil para notificar de la cancelación de las tareas con la máxima brevedad posible.
- *Métodos*:
  - *obtenerNColegas*: retorna el valor del atributo *numColegas*.
  - *obtenerContacto*: retorna el teléfono correspondiente a uno de los colegas integrantes del equipo.
  - *materializar*: transforma la información proveniente de una fuente de almacenamiento persistente a un objeto de la clase.

Clase *informcPropia*

Esta clase contiene una serie de información relativa al propio usuario, su rol y también los medios disponibles para ponerse en contacto con él, como son el e-mail, número de teléfono, URL, etc., a fin de poder automatizar ciertas acciones y facilitar la información necesaria para establecer cualquier tipo de contacto con el mismo. Así por ejemplo, la información del rol permite filtrar la realización de ciertas acciones.

Por supuesto, proporciona los métodos necesarios para acceder a toda esta información.

En el diagrama de clase 8.16 del *Capítulo 8* (véase *Capítulo 8*; *sección 8.2*), correspondiente a la versión proporcionada por el FPI, quedan reflejados todos estos detalles. Dado que puede ser utilizada por más de una clase, se ha decidido diseñarla como un *Singleton*, a fin de garantizar una única instancia.

B.3.2. Capa aspectual

Los *aspectos* que se proponen en la versión genérica para apoyar el trabajo en grupo corresponden a las tres metas a abordar en un entorno colaborativo, identificadas por Ellis en [EGR91] : colaboración, coordinación y comunicación. Se destina un *aspecto* para cada una de ellas.

Aspecto Coordinación

- *Objetivo*: monitorizar las incidencias en las que la incorporación de una serie de acciones adicionales puede ser de gran ayuda en lo que respecta a la coordinación entre los miembros del grupo de trabajo.

  Así, en el caso de estudio trabajado se considera adecuado controlar la cancelación de tareas por parte de uno de los miembros, a fin de avisar y poner al corriente al resto de los miembros implicados en dicha tarea.

- *Atributos*:

  - *consciencia*: una referencia a la única instancia de la clase *cGrupoParticular*.

- *Puntos de corte*:

  - *comunicarTareaCancelada*.

    - *puntos de unión*: se trata de interceptar la ejecución de la cancelación de una tarea por parte del usuario, como consecuencia de haberse producido cierta incidencia que impide su desempeño tal y como estaba planificado. El *punto de unión* captura la invocación al método encargado de marcar una tarea como cancelada. Además, captura el objeto invocado a través de *target* (véase *Apéndice A*), que en este caso es un objeto de la clase *Tarea*, necesario para proceder a la notificación al resto de componentes del equipo.
    - *consejo asociado*: una vez finalizada la ejecución del método capturado, se procede a comunicar vía SMS a todos y cada uno de los miembros implicados (método *enviarAvisosSMS*). Esta información se obtiene precisamente

a través del uso de la clase *EquipoTarea* que se incorpora como parte del *modelo contextual*. A continuación se procede a actualizar la *consciencia* y finalmente se le comunica la incidencia al *servidor de plasticidad*, para lo cual se requiere serializar la información relativa a la tarea, así como recurrir a parte de la información almacenada en la instancia de la clase *informcPropia* (método *enviarServidorTareaCancelada*).

- *recibirNotificaciónTareaCancelada*.
  - *puntos de unión*: se intercepta la ejecución del método que se encarga de tratar la recepción de un mensaje SMS, siempre y cuando se trate de una notificación de cancelación de tarea.
  - *consejo asociado*: Se captura el mensaje recibido a fin de conocer de qué tarea se trata (método *extraerTareaDeSMS*) y poder proceder de manera automática a la cancelación de la misma (método *CancelacionAspectualTarea*). A continuación se procede a actualizar la *consciencia*.

Aspecto Comunicación

- *Objetivo*: monitorizar las incidencias en las que una adecuada y oportuna comunicación con alguna de las partes implicadas en la actividad grupal puede ser de gran ayuda en el desarrollo del trabajo conjunto.

  En el caso de estudio trabajado se considera adecuado controlar la situación en que el stock de un determinado material se pone por debajo de un determinado umbral, a fin de avisar y poner al corriente de la situación al departamento de administración.

- *Atributos*:
  - *consciencia*: una referencia a la única instancia de la clase *cGrupoParticular*.

- *Puntos de corte*:
  - *detectarStockBajo*.
    - *puntos de unión*: se intercepta la ejecución del método encargado de actualizar las existencias en stock de los materiales, y se intercede cuando se produce la circunstancia de tener el nivel de stock por debajo de un cierto umbral. Además, captura el objeto propietario a través de *this* (véase *Apéndice A*).
    - *consejo asociado*: actualizar la consciencia y proceder a comunicar acerca de la incidencia al departamento de administración vía e-mail (método *enviarEmailAdmn*).

Aspecto Colaboración

- *Objetivo*: monitorizar las circunstancias en que se produce un avance en la actividad a desarrollar por parte del usuario, o cualquier otra situación suficientemente significativa, de cara a obtener una perspectiva global de la situación del trabajo en grupo por parte del *servidor de plasticidad*. En este caso se ha tenido en cuenta tanto la circunstancia de finalización de una tarea, como la de completitud de la agenda de la jornada.
- *Atributos*:
  - *consciencia*: una referencia a la única instancia de la clase *cGrupoParticular*.
- *Puntos de corte*:
  - *notificarFinTarea*.
    - *puntos de unión*: se intercepta la ejecución del método encargado de marcar una tarea como acabada. Además, captura el objeto propietario del método a través de *this* (véase *Apéndice A*), que en este caso es un objeto de la clase *Tarea*, necesario para proceder a la notificación de la circunstancia al *servidor de plasticidad*.
    - *consejo asociado*: una vez finalizada la ejecución del método capturado, se procede a comunicarle al *servidor de plasticidad* de la circunstancia vía una petición, siempre y cuando se trate de un usuario que desempeña el papel de encargado –con objeto de evitar redundancias en las notificaciones (método *enviarServidorTareaAcabada*). A continuación se procede a actualizar la *consciencia*.
  - *notificarFinActividad*.
    - *puntos de unión*: se intercepta la circunstancia en la que el usuario completa toda la actividad que se le había planificado. Se trata de comprobar si el atributo *numTareasPendientes* de la clase *cGrupoParticular* es o no igual a cero (*punto de unión* condicional).
    - *consejo asociado*: una vez detectada la circunstancia se procede a actualizar la *consciencia* y finalmente a comunicarle la nueva situación al *servidor de plasticidad* (método *avisarServidorEstadoPasivo*). En efecto, se trata de una situación de trascendencia para la construcción del *conocimiento compartido*, dado que un miembro ha variado su estado en relación a la actividad grupal. Además de la notificación pertinente se le envía también

toda la información recopilada en la *consciencia*, para lo cual se requiere su serialización (método *serializarXML* de la clase *cGrupoParticular*).

En los diagramas de clase 8.17 y 8.18 de la *sección 8.3*, correspondientes a las versiones proporcionadas por el FPI, quedan reflejados todos estos detalles.

B.4. Componente genérica de adaptación al entorno o recurso hardware

El propósito de este objetivo de personalización se describe en detalle en el *Capítulo 7* (véase *Capítulo 7*; *sección 7.1.3.3.1.* y *7.2.3.2.1.*).

B.4.1. Capa sensible al contexto

Esta capa contiene toda la funcionalidad descrita al respecto del tratamiento de un determinado factor ambiental. Con objeto de facilitar la comprensión, legibilidad, reutilización y mantenimiento del código correspondiente a esta capa se opta por descomponer el diseño de la *capa sensible al contexto* en diversas piezas de código, con objeto de aumentar la cohesión y clarificar los objetivos de cada una de ellas. Se han aplicado los patrones GRASP (General Responsibility Assignment Software) [Lar03] *alta cohesión*, *controlador* y *experto* para este propósito. Las piezas de código obtenidas son las siguientes:

Clase *GestorFactorExterno*

- *Objetivo*: Articula la funcionalidad de la *capa sensible al contexto* actuando como interfaz de comunicación con la *capa aspectual*. De hecho, encapsula los objetos principales de la *capa sensible al contexto*, los cuales son manipulados por una instancia de esta clase. Para asegurar la coexistencia de una única instancia (en ocasiones es accedida también por un *thread* –véase el caso de estudio 'Lector de Noticias; *Capítulo 7 - sección 7.2.3.2.1.*) se le aplica el patrón GoF *Singleton* [GHJV95].

  Sus responsabilidades son: (1) la creación del sensor y de la clase que representa el histórico de los cambios efectivos en la *restricción de tiempo real* observada; (2) la construcción, mantenimiento y envío al *servidor de plasticidad* del histórico recogido; (3) establecer la comunicación con el sensor, a fin de obtener los sucesivos valores percibidos en la *restricción de tiempo real* cada vez que son requeridos por parte de la unidad de programa *aspecto* (y en su caso también por el *thread* paralelo).

  En definitiva, tiene asignada una responsabilidad de controlador.

- *Atributos*:
  - *nivelFactor*: recoge el valor captado por el sensor en el momento de realizar una consulta al mismo.
  - *lista*: una instancia de la estructura de datos encapsulada por la clase *HistoricoCambios*, la cual contiene la relación de cambios en el factor ambiental que han dado lugar a una adaptación en la IU.
  - *instancia*: atributo estático para referenciar la única instancia de la clase.
- *Métodos*:
  - *método constructor*: inicializa todos los objetos de la *capa sensible al contexto* que encapsula. Además, al tratarse de un *Singleton*, viene provista del método de obtención de su única instancia, el método *obtenerGestor*.
  - *AdquirirValorSensor*: procede a realizar una lectura del valor captado por el sensor. Como en ocasiones se recurre a un *thread* paralelo, encargado también de efectuar consultas al sensor, este método está protegido con exclusión mutua para evitar conflictos de concurrencia.
  - *anotarCambio*: introduce un nuevo valor en el histórico de cambios a registrar. Este método se invoca tan sólo cuando el cambio producido se considera suficientemente significativo, esto es, si ha provocado una adaptación.
  - *cursarHistorico*: se encarga de completar la información del histórico con la fecha de fin de sesión, serializar todos los valores recogidos durante la sesión y finalmente de enviarlos al servidor.

Clase *HistoricoCambios*

- *Objetivo*: soportar en memoria la estructura de datos para almacenar los valores del sensor que han ocasionado una adaptación a lo largo de la sesión, proporcionando la operativa necesaria para su correcta manipulación.
- *Atributos*:
  - *valores*: es la colección de los valores del sensor a ser registrados en el histórico.
  - *fechaInicio*: día y hora de inicio de la sesión.
  - *fechaFin*: día y hora de finalización de la sesión.
- *Métodos*:
  - *método constructor*: inicializa la estructura de datos utilizada.

- *introducirCambio*: añade un nuevo valor a la *lista*.
- *Datar*: asigna la fecha y hora de finalización de la sesión.
- *serializarXML*: serializa todos los valores recogidos en formato XML.
- *serializar*: convertir la instancia de la clase al formato adecuado para su almacenamiento persistente.
- *materializar*: traducir a objetos de programa los valores del histórico almacenados de forma persistente.

Interfaz *adaptadorSegunFactor*

La acción de regular la pantalla en un teléfono móvil generalmente está supeditada a la marca del dispositivo, por lo que es necesario recurrir a la API propietaria para acceder al método específico que se encarga de aplicar el efecto buscado en ese tipo de dispositivos. En consecuencia, la parte correspondiente a la ejecución de la adaptación interesa que esté perfectamente identificada y encapsulada. De ese modo, y conociendo el método en cuestión que contiene esa variación, no es necesario cambiar la parte de invocación al mismo. Además, puede procederse fácilmente a su especialización para un caso concreto sin que ello afecte al resto de código.

En este sentido, es oportuno incluir una componente del tipo interfaz que declare el método encargado de encapsular esa adaptación, a fin de ser implementado por la clase *ReguladorPantalla*. El método ofrecido se denomina *aplicarAdaptacion*. Esta estrategia evita tener que aplicar variaciones generalizadas en las clases y *aspectos* de esta componente de adaptación, siendo necesario tan sólo modificar el código de un método.

Este método encapsula el tratamiento a procesar tanto en el caso en que la adaptación al factor ambiental consiste en aplicar un determinado efecto sobre la pantalla –uso de las APIs propietarias-, como cuando se requiere realizar otro tipo de operaciones más complejas, destinadas a regular o decidir la modalidad como llevar a cabo una determinada funcionalidad u operación en determinados casos.

Clase *ReguladorPantalla*

Esta es la clase que implementa la interfaz *adaptadorSegunFactor*. Su responsabilidad es, por tanto, la de especificar la manera como llevar a cabo la adaptación ante un cambio en el factor contextual objeto de estudio (implementación del método *aplicarAdaptación*), que es particular a cada caso. Dependiendo de su complejidad, en ocasiones convendrá descomponerlo en una serie de métodos privados, los cuales serían también estáticos.

Para evitar alteraciones en otros puntos de la componente de adaptación al entorno, es importante implementarla manteniendo el nombre de la clase propuesto en el framework, dado que éste es utilizado por el aspecto *ControlFactorAmbiental*.

Todo este tipo de funcionalidades auxiliares destinadas a aplicar efectos sobre la pantalla o a regular vía código el funcionamiento de determinados parámetros del dispositivo (e. g. el volumen de una reproducción), no son satisfechas por la API básica de MIDP 2.0. Se esperan incluir en la incipiente MIDP 3.0[2]. Es necesario recurrir al uso de paquetes opcionales, que para que sean operativos deben ir incorporados en los teléfonos móviles utilizados. En concreto, la funcionalidad de regular el brillo de la pantalla la ofrecen los teléfonos Nokia de las series 40 y 60[3]. Aún así, como las APIs propietarias son paquetes definidos por un determinado fabricante, no siempre siguen las especificaciones estándar, ni tampoco cumplen el *test de compatibilidad*[4]. En consecuencia, en determinados casos no acaban ofreciendo la funcionalidad especificada. Esa es una de las repercusiones del problema conocido como *fragmentación de APIs* (véase *glosario*). En particular, éste ha sido uno de los problemas encontrados en los casos de estudio relacionados (véase *Capítulo 7*), experimentación llevada a cabo con la familia de teléfonos móviles mencionada. A pesar de venir provistos de la funcionalidad deseada, ésta no es operativa.

Interfaz *perceptorEntorno*

Tampoco la acción de capturar el valor de un sensor es satisfecha por la API básica de MIDP 2.0. Es necesario recurrir de nuevo al uso de paquetes opcionales que resuelvan la comunicación con aquél, que para que sean operativos deben estar incorporados en los teléfonos móviles utilizados.

Java Community Process (JCP) ha lanzado la especificación JSR 256, denominada *Mobile Sensor API*[5], la cual define una API genérica para las aplicaciones J2ME que permite obtener datos tanto de sensores empotrados, como de sensores conectados vía infrarrojos, Bluetooth o GPRS de forma fácil y genérica. No obstante, existen dos aspectos a tener en cuenta:

1. Si se piensa en una componente genérica para el tratamiento del entorno y de las restricciones hardware, esto es, válida para cualquier tipo de factor contextual, la parte

[2]Mobile Information Device Profile 3. Especificación JSR 271 programada para el 2007. Consúltese on-line en: http://jcp.org/en/jsr/detail?id=271

[3]Consúltese on-line en: http://www.forum.nokia.com/info/sw.nokia.com/id/bceaffad-1807-43b6-9cd9-8519b4794b5c/S60_Platform_Basics_v1_0_en.pdf.html

[4]Juego de pruebas que permite verificar si una determinada implementación cumple o no la especificación JSR a la que pertenece.

[5]Consúltese on-line en: http://forum.nokia.com/info/sw.nokia.com/id/f21597a3-ace5-4746-b57b-896047d4c04d.html

encargada de comunicarse con el sensor deberá estar suficientemente encapsulada e identificada.

2. Por otro lado, en el momento de realizar las pruebas no existían dispositivos móviles que llevaran la especificación JSR 256 incorporada, y por tanto la comunicación de las aplicaciones J2ME con los sensores del dispositivo no es todavía factible. Efectivamente, en ocasiones aparecen especificaciones estándar para dar solución a ciertas funcionalidades sin que exista ningún dispositivo que las implemente. Esta es otra de las dificultades con las que nos hemos encontrado en el momento de validar sobre móviles reales la implementación de los casos de estudio relacionados con la luminosidad ambiental (véase *Capítulo 7*).

   Por consiguiente, el módulo correspondiente para los dos casos de estudio lo que hace es simular la lectura de valores del sensor.

   No obstante, esta distinción no afecta a las clases que utilizan este método, puesto que aparece encapsulado, de tal forma que el modo como llevar a cabo la adquisición de los distintos valores es transparente. En definitiva, el día en que esta funcionalidad esté operativa, su puesta en marcha no supondrá tener que aplicar cambio alguno en la componente genérica aquí descrita.

Ambos motivos llevan a pensar que lo más conveniente es ofrecer una interfaz, de manera que la parte correspondiente a la lectura del sensor esté perfectamente identificada y el código asociado apropiadamente encapsulado. De ese modo, conociendo el método encargado de esta parte no es necesario cambiar los puntos del código que contienen la invocación del mismo. El método ofrecido se denomina *obtenerValorSensor*, el cual es implementado por la clase denominada *Sensor*.

Clase *Sensor*

Esta es la clase que implementa la interfaz *perceptorEntorno*, y por tanto es la encargada de particularizar la manera como llevar a cabo la comunicación y consulta del sensor correspondiente a un determinado factor ambiental, implementando el método *obtenerValorSensor* ofrecido por dicha interfaz.

La clase *Sensor* puede incluir un método constructor no vacío. Generalmente se requiere incluir código de inicialización cuando se trata de una simulación, y por tanto es necesario preparar la fuente de los datos. Esta clase se define aplicando el patón GoF *Singleton*.

De acuerdo a lo mencionado, a pesar de que esta clase deba ser implementada para cada caso particular, respetando el nombre y signaturas de los métodos propuestos, la

línea de código correspondiente a la consulta del sensor no varía, lo que proporciona la reutilización también de la clase cliente –en nuestro caso la clase *GestorFactorExterno*.

Clase *ThreadSimple*

Esta clase encapsula un hilo de ejecución (un *thread*) encargado de automatizar las consultas al sensor deseado cada cierto tiempo. Se incorpora tan sólo en aquellas aplicaciones donde se requiere complementar la actuación de la unidad *aspecto* descrita en la sección siguiente con una comprobación en paralelo del valor del sensor. Generalmente es adecuada cuando existe la posibilidad de que el usuario permanezca periodos de tiempo relativamente largos consultando la pantalla sin realizar ningún tipo de interacción, como en el caso de estudio *'Lector de Noticias'* (véase *Capítulo 7*; *sección 7.2.3.2.1.*), pensando en los periodos en que el usuario procede a leer una noticia en pantalla.

En el diagrama de clase 8.19 del *Capítulo 8* (véase *Capítulo 8*; *sección 8.2*), correspondiente a la versión proporcionada por el FPI, quedan reflejados todos estos detalles.

B.4.2. Capa aspectual

Se propone un único *aspecto* para acoplar las *capas lógica* y *sensible al contexto*, el cual automatiza la consulta y control del factor ambiental, así como la consecuente regulación de la IU o adaptación funcional.

Aspecto ControlFactorAmbiental

- *Objetivo*: el objetivo de esta clase es múltiple: disparar las oportunas consultas al sensor del factor contextual en cuestión, detectar los cambios producidos, decidir cuándo esos cambios deben ser reflejados en la IU, y llevar a cabo la oportuna adaptación, ya sea regulando un parámetro de la pantalla o bien alterando la manera como llevar a cabo una funcionalidad concreta, cuya ejecución puede ser crítica dependiendo del estado de ciertos recursos hardware.
- *Atributos*:
  - *valorAnterior*: mantiene el último valor del factor ambiental que provocó la ejecución de una adaptación o regulación de la pantalla.
  - *gestor*: una referencia a la única instancia de la clase *GestorFactorExterno*, a efectos de controlar el sensor y generar el histórico.
- *Puntos de corte*:

- *controlarCambios*.
  - *puntos de unión*: se monitorizan los cambios en el valor de luminosidad, valor representado en la clase controladora *GestorFactorExterno* a través de un atributo. Este tipo de *puntos de unión* se expresan con *set* en el lenguaje *AspectJ* (véase *Apéndice A*). Por otro lado, este *punto de corte* incluye la condición que restringe cuándo un cambio en el factor ambiental será considerado un cambio suficientemente significativo como para aplicar la adaptación.
  - *consejo asociado*: lleva a cabo la acción encargada de regular la IU de acuerdo al cambio detectado, así como de activar la operación encargada de registrar ese nuevo valor en el histórico. Asimismo, actualiza el valor del atributo *valorAnterior*.
- *ActivarConsultaSensor*.
  - *puntos de unión*: se capturan todos los métodos de la aplicación base encargados del tratamiento de las interrupciones provocadas por la propia interacción del usuario con la IU[6] o bien, dependiendo de la *restricción de tiempo real* a controlar y el tipo de aplicación, se intercepta la ejecución de la funcionalidad considerada crítica.
  - *consejo asociado*: activa la consulta al sensor del factor ambiental, a través de la controladora *GestorFactorExterno*. En ocasiones la adaptación a aplicar se encuentra encapsulada también en este *consejo* (método *discernirValorSensor*).
- *CerrarHistórico*.
  - *puntos de unión*: intercepta la ejecución de un método encargado del cierre de la ejecución.
  - *consejo asociado*: invoca el método de la clase *GestorFactorExterno* encargado de cursar el histórico de cambios (invocación al método *cursarHistórico* de esta clase).

En los diagramas de clase 8.21 y 8.21 del *Capítulo 8* (véase *Capítulo 8*; *sección 8.2*), correspondientes a las versiones proporcionadas por el FPI, quedan reflejados todos estos detalles.

[6]Los métodos *commandAction* y *ItemStateChanged* en la API para la interfaz de usuario de alto nivel en J2ME.

Apéndice C

Diagramas complementarios del diseño del *Framework de Plasticidad Implícita*

En este apéndice se recogen los diagramas de clases que representan cada uno de los componentes desarrollados del *Framework de Plasticidad Implícita*, donde también queda plasmado su impacto sobre la aplicación base (*capa lógica*). En consecuencia, representan el *Motor de Plasticidad Implícita* resultante de acoplar los componentes en una aplicación concreta.

Estos diagramas complementan los que se encuentran en el *Capítulo 8*, los cuales han ido mostrando la estructura de ambas capas del framework (*capas sensible al contexto* y *aspectual*) por separado. Por lo tanto, esta perspectiva conjunta del *Motor de Plasticidad Implícita* permite apreciar las interrelaciones entre las distintas piezas de código entre capas.

En total son cinco los componentes desarrollados: (1) componente de personalización de pantalla para objetivo A (valores por defecto en un formulario); (2) componente de personalización de pantalla para objetivo B (ordenación de una lista); (3) componente de personalización de una funcionalidad concreta del sistema base; (3) componente de apoyo al trabajo en grupo y (5) componente de adaptación al entorno o restricción en recurso hardware. Al igual que en el *Capítulo 8*, se presentan las dos versiones desarrolladas: para aplicaciones fijas construidas en J2SE y para aplicaciones móviles construidas en J2ME. En este último caso, cuando la componente es específica para el perfil MIDP, como perfil particular dentro de la modalidad J2ME, así queda plasmado en el diagrama.

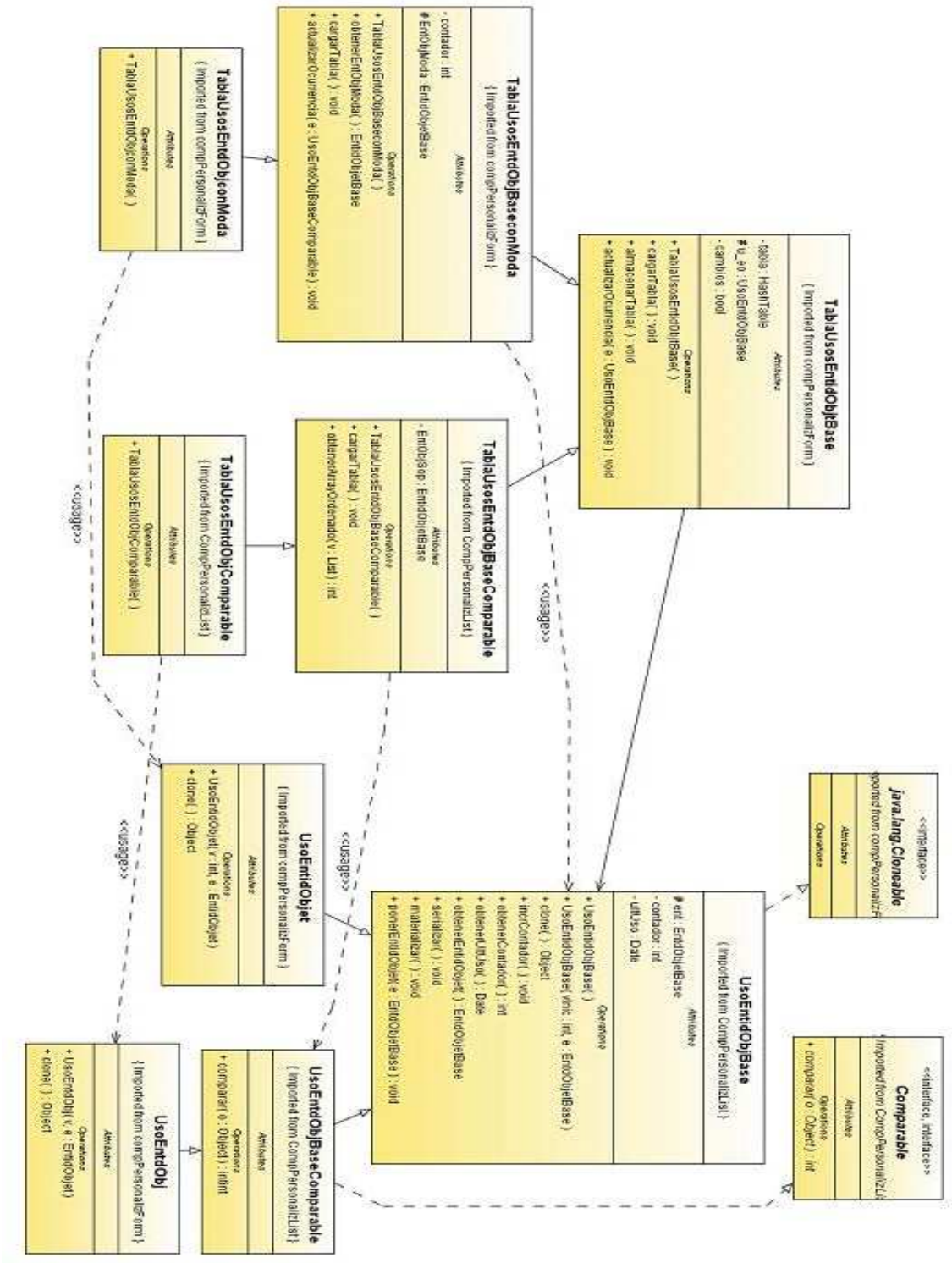


Figura C.1: Interrelaciones entre las jerarquías de tablas y usos (capa SC) en la construcción de componentes de personalización de pantalla.

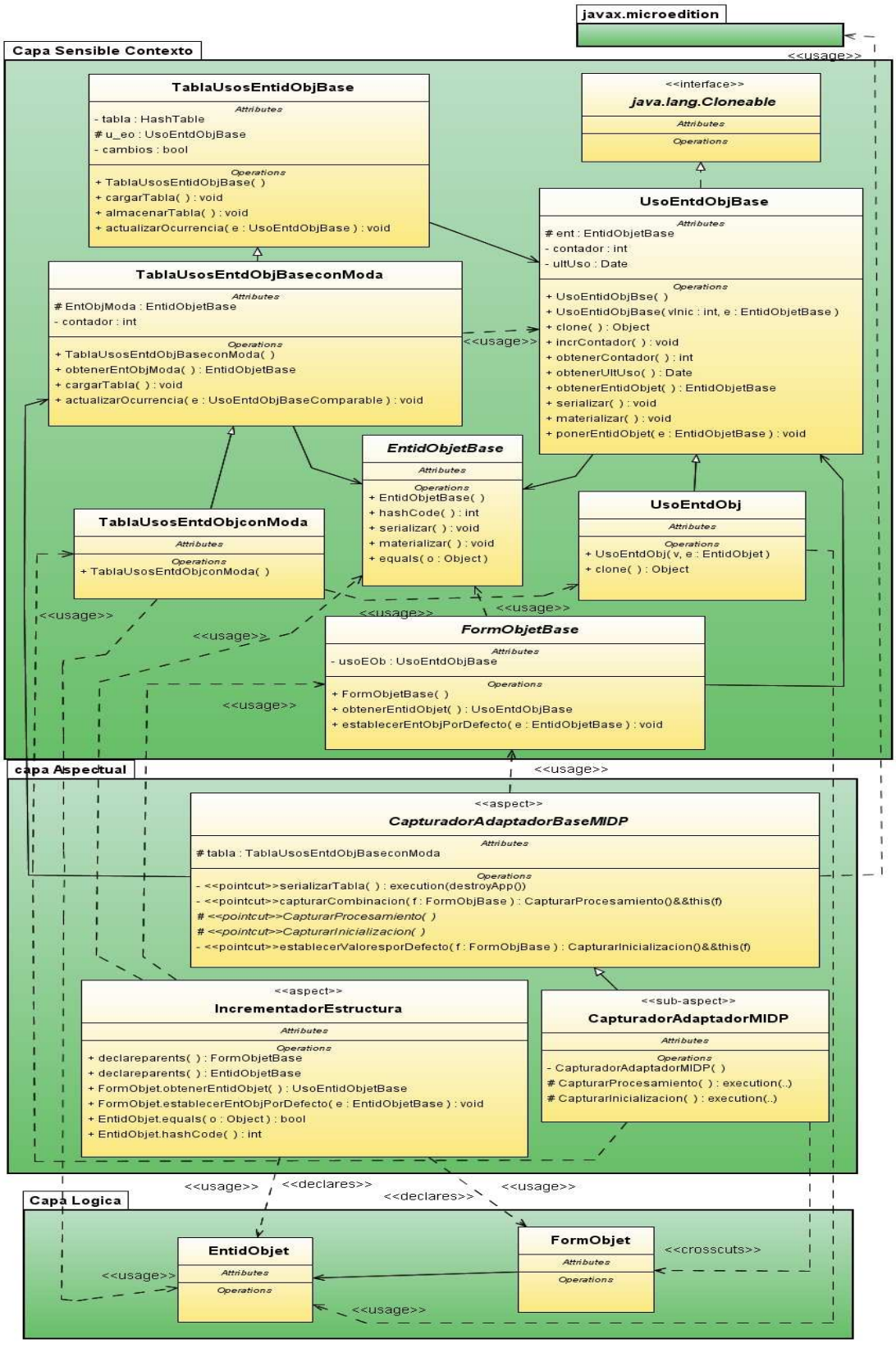


Figura C.2: Componente de personalización de pantalla para el perfil MIDP - Objetivo A.

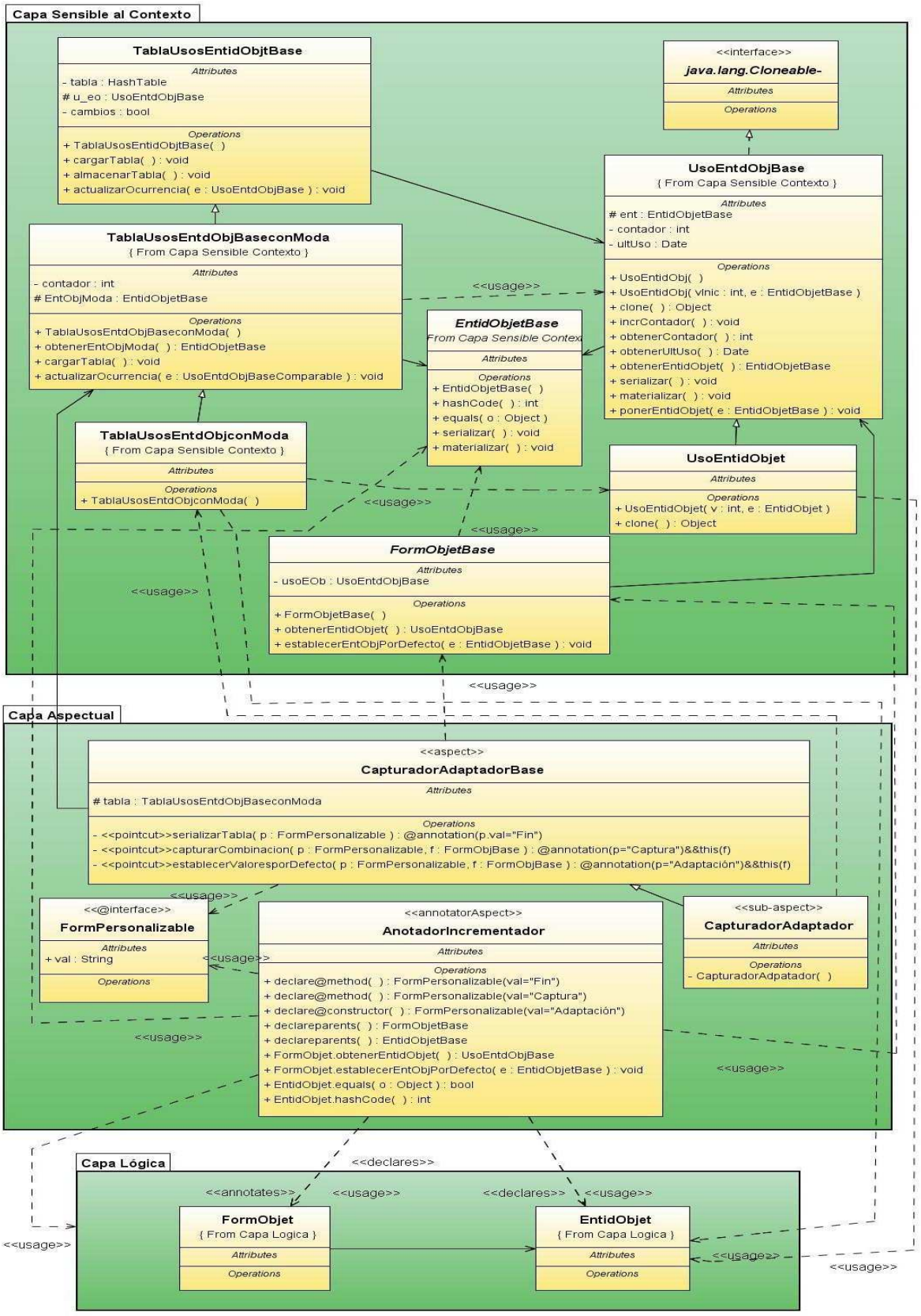


Figura C.3: Componente de personalización de pantalla para J2SE - Objetivo A.

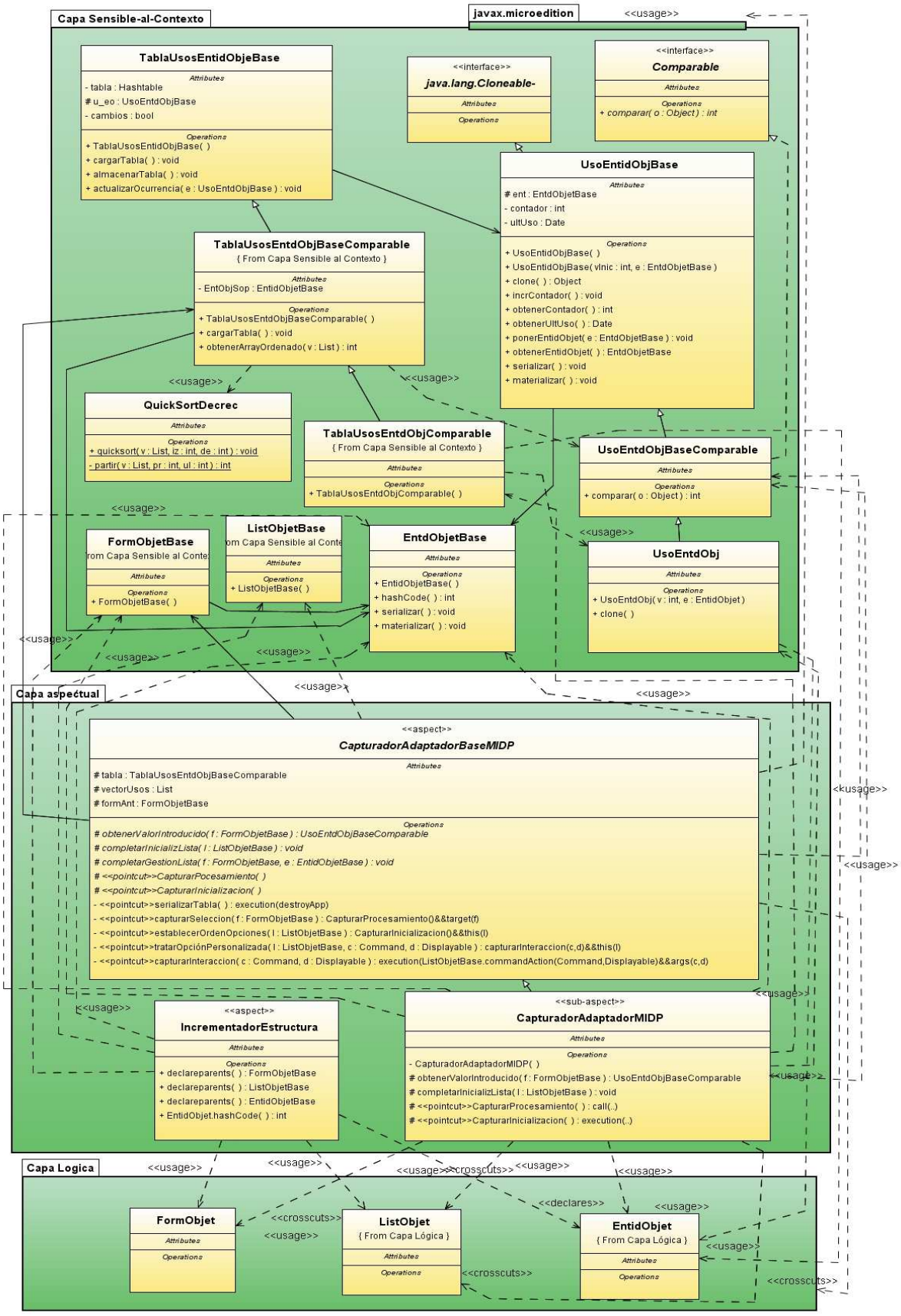


Figura C.4: Componente de personalización de pantalla para el perfil MIDP - Objetivo B.

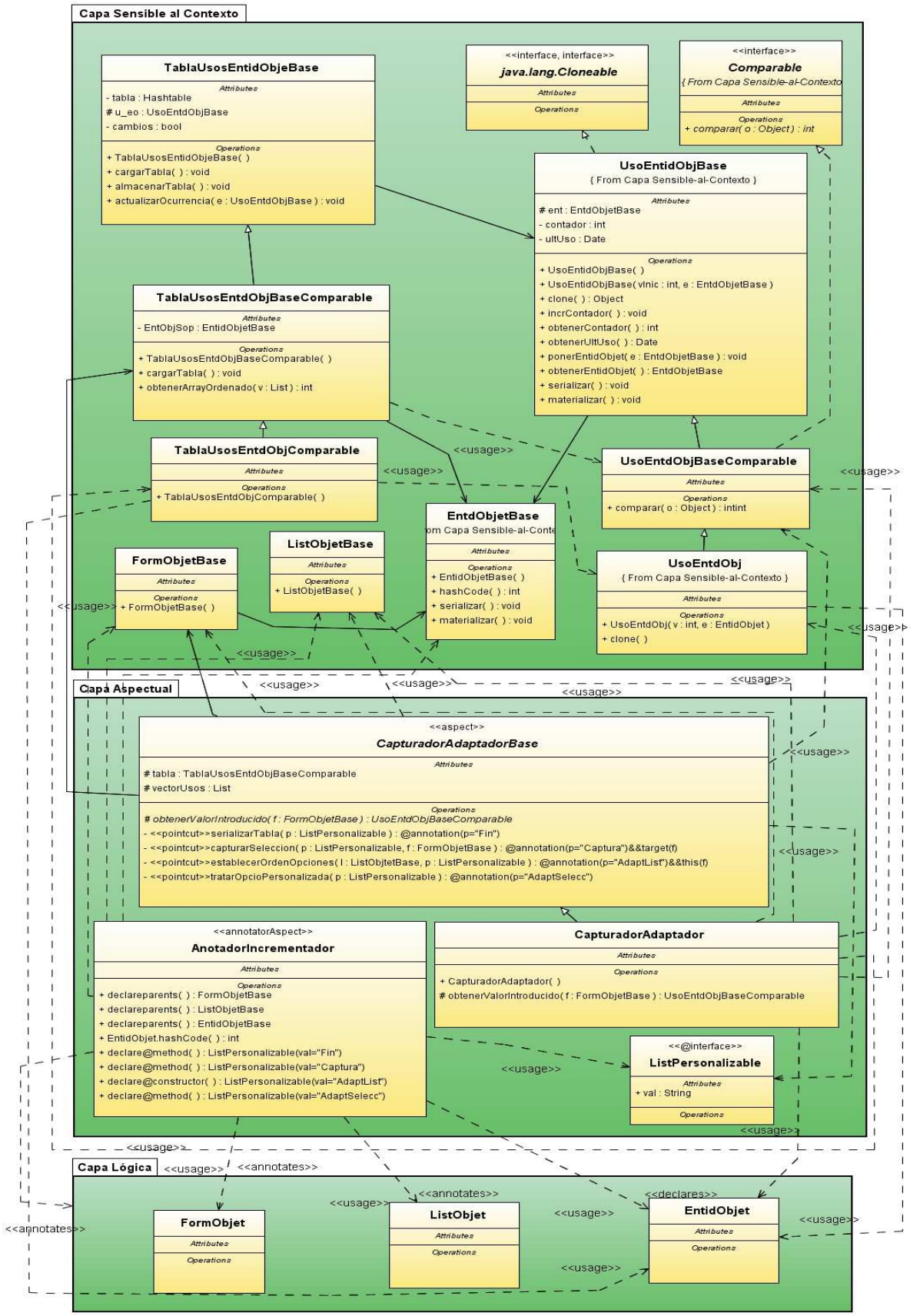


Figura C.5: Componente de personalización de pantalla para J2SE - Objetivo B.

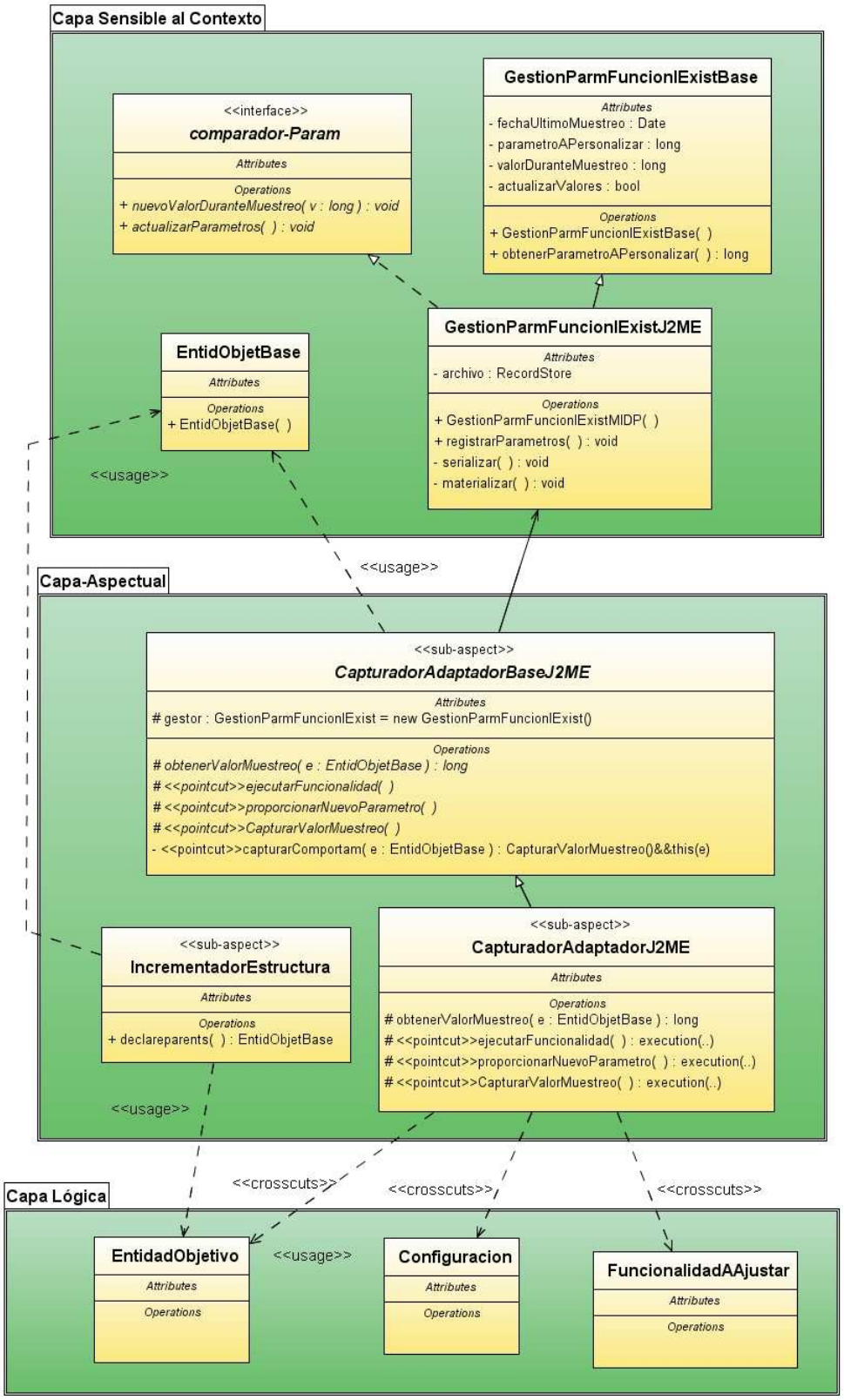


Figura C.6: Componente de personalización de una funcionalidad concreta para J2ME.

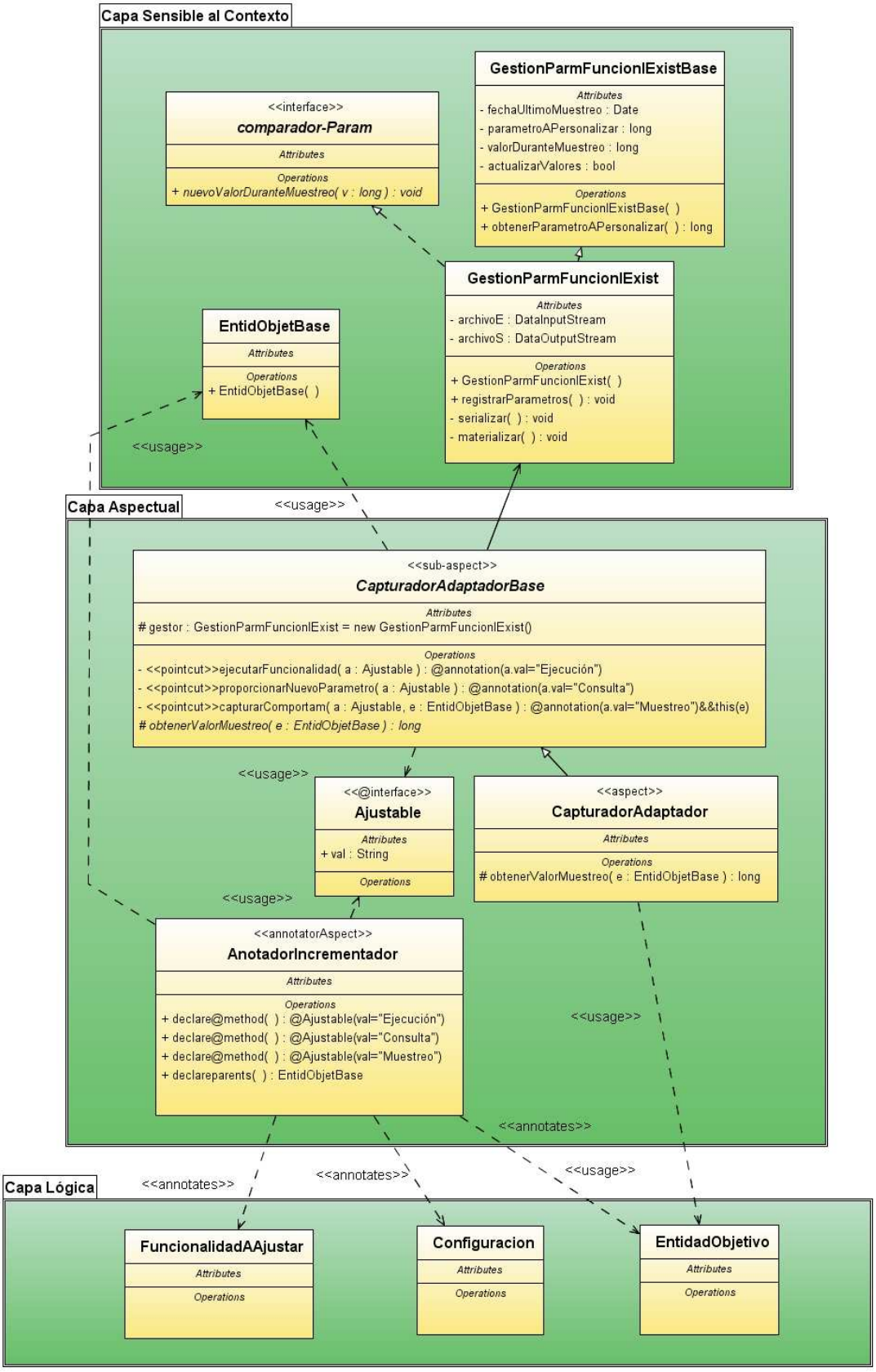


Figura C.7: Componente de personalización de una funcionalidad concreta para J2SE.

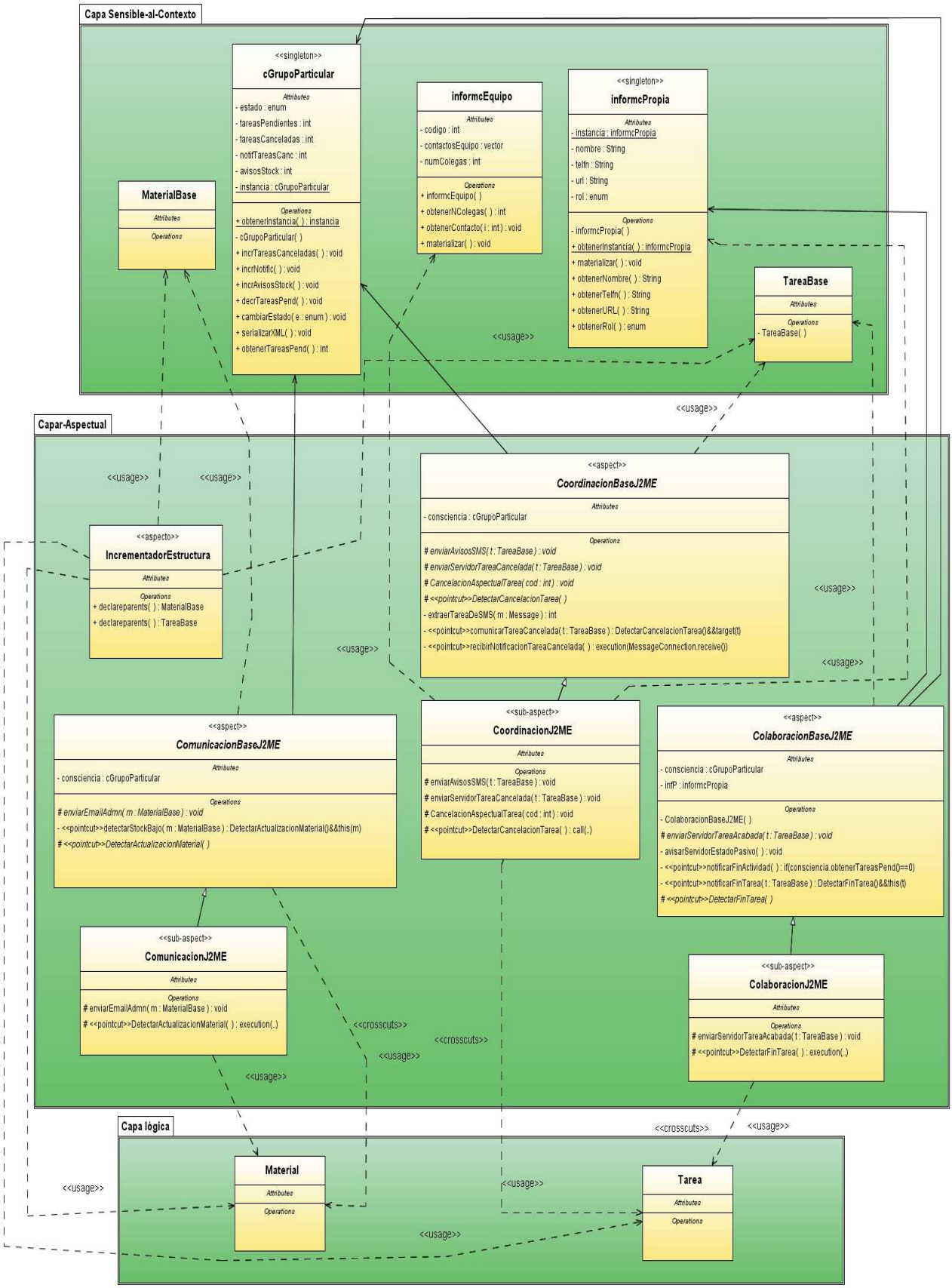


Figura C.8: Componente de apoyo al trabajo en grupo para J2ME.

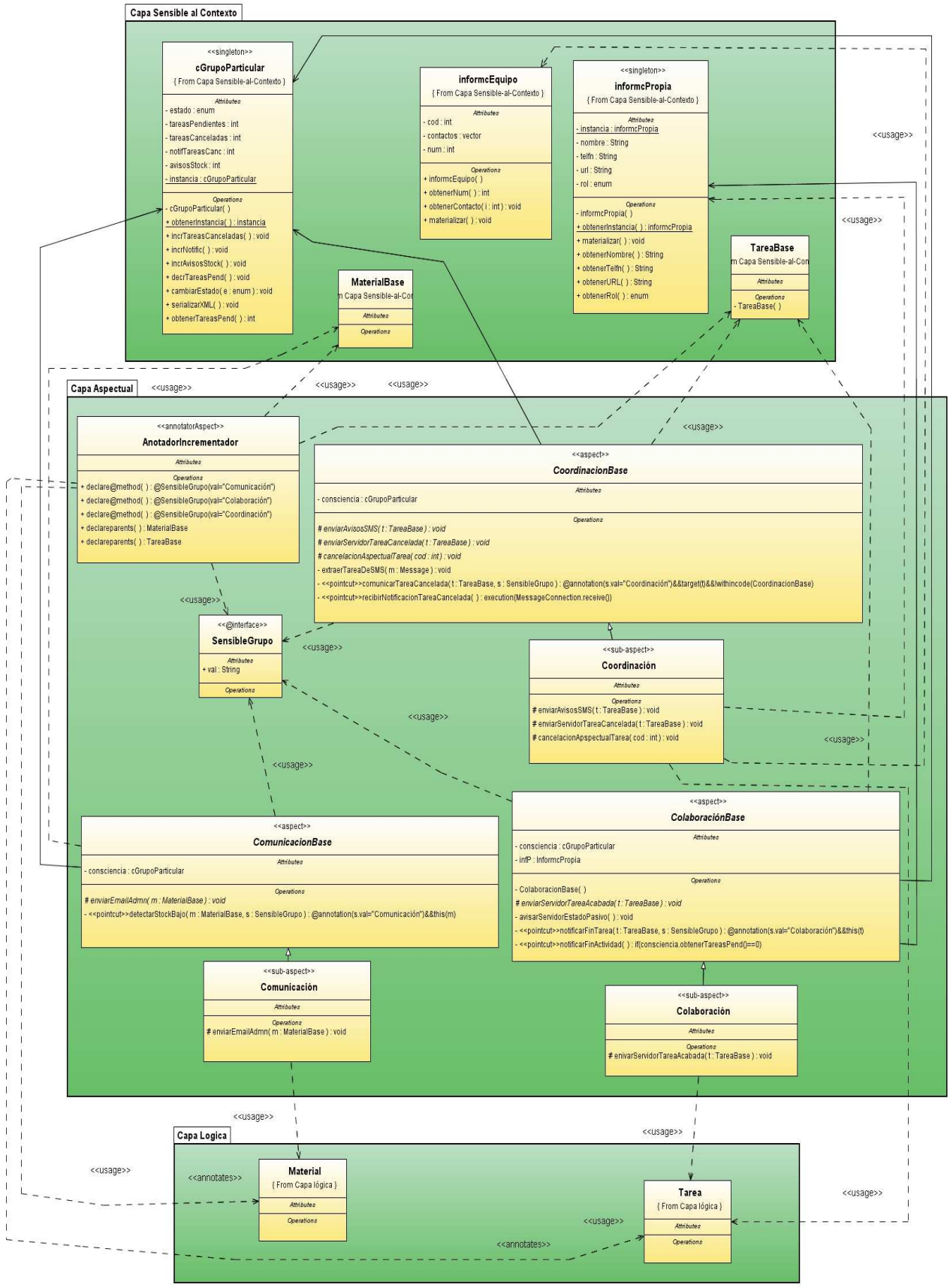


Figura C.9: Componente de apoyo al trabajo en grupo para J2SE.

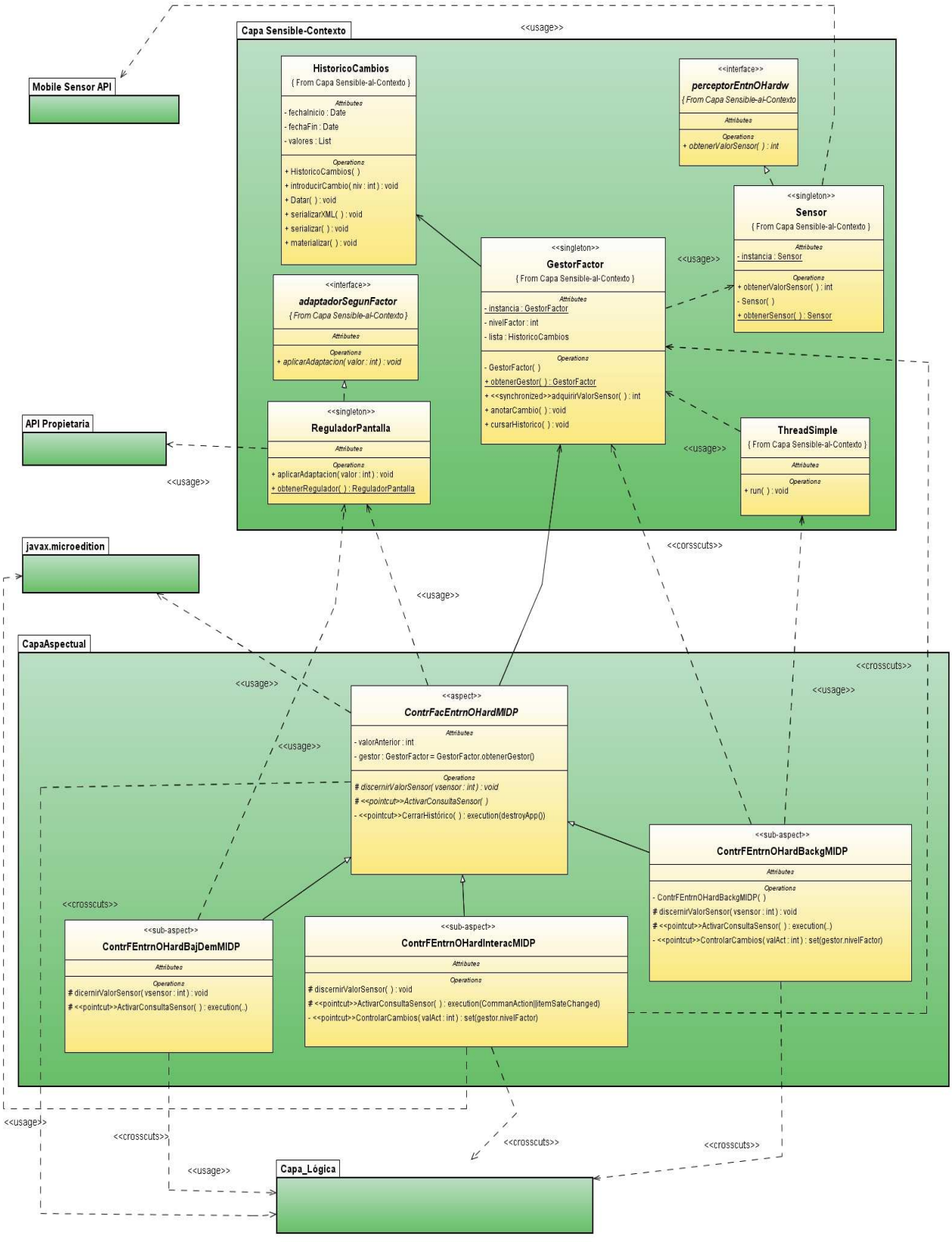


Figura C.10: Componente de adaptación al entorno o restricción en recurso hardware para el perfil MIDP.

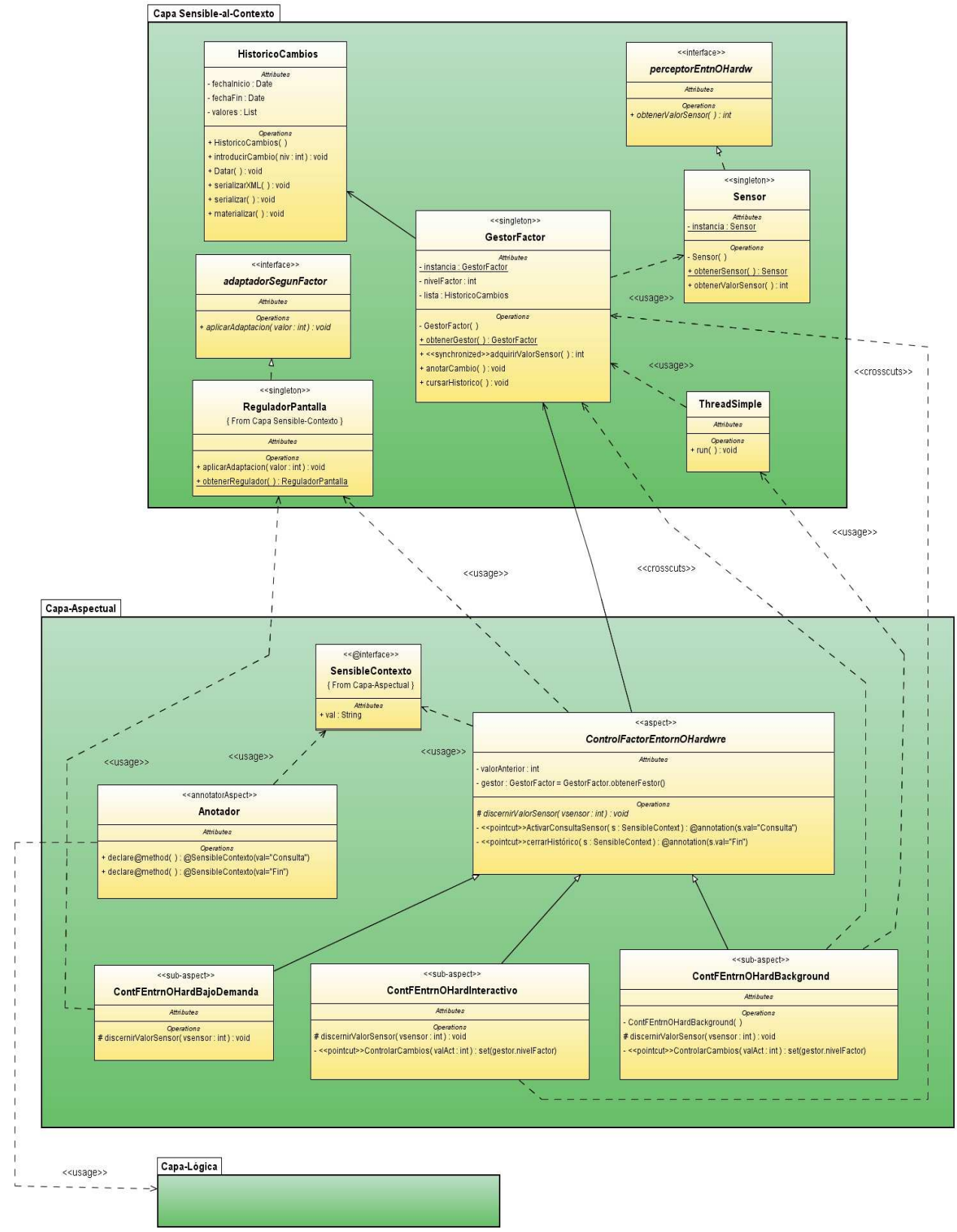


Figura C.11: Componente de adaptación al entorno o restricción en recurso hardware para J2SE.

Bibliografía

[AAC+02] J. Abascal, I. Aedo, J.J. Cañas, M. Gea, A.B. Gil, J. Lorés, A. Martínez, M. Ortega, P. Valero, and M. Vélez. Introducción a la Interacción Persona-Ordenador. Asociación Interacción Persona-Ordenador (AIPO). ISBN: 84-607-2255-4, 2002. J. Lorés (Ed.).

[AAH+97] G.D. Abowd, C.G. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton. Cyberguide: a mobile context-aware tour guide. *ACM Wireless Networks*, 3(5):421–433, 1997.

[ABV92] M. Aksit, L. Bergmans, and S. Vural. An object-oriented language-database integration model: The composition-filters approach. In O. Lehrmann Madsen, editor, *Proc. of the 7th. European Conference on Object-Oriented Programming (ECOOP'92)*, pages 372–395, London, UK, 1992. Springer-Verlag. ISBN: 3-540-55668-0.

[ACK94] A Asthana, M Cravatts, and P Krzyzanowski. An indoor wireless system for personalized shopping assistance. In M. Satyanarayanan, editor, *Proc. of IEEE Workshop on Mobile Computing Systems and Applications*, volume 29, pages 69–74, New York, NY, USA, 1994. ACM Press. ISBN: 0-8186-6345-6.

[Ada95] D. Adair. Building object-oriented frameworks - parts 1 and 2. Taligent White paper, 1995. AIXpert.

[AFJP04] M. Amor, L. Fuentes, D. Jiménez, and M. Pinto. Adaptive collaborative virtual environments: A component and aspect-based approach. *Inteligencia Artificial. Revista Iberoamericana de IA, publicación de la Asociación Española para la Inteligencia Artificial (AEPIA)*, (24):33–43, 2004.

[AG97] R. Allen and D. Garlan. A Formal Basis for Architectural Connection. *ACM Transactions on Software Engineering and Methodology*, 6(3):213–249, 1997.

[AGOP05] R.A. Alarcón, L.A. Guerrero, S.F. Ochoa, and J.A. Pino. Context in Collaborative Mobile Scenarios. In *Workshop on Context and Groupware (CONTEXT'05)Fifth International and Interdisciplinary Conference on Modeling and Using Context*, volume 133, Paris, France, 2005. CEUR Proc.

[AH95] Y. Arens and E.H. Hovy. The Design of a Model-Based Multimedia Interaction Manager. *Artificial Intelligence Review*, 9(2–3):167–188, 1995.

[AM00] G.D. Abowd and E.D. Mynatt. Charting past, present, and future research in Ubiquitous Computing. *ACM Transactions of Computer-Human Interaction*, 7(1):29–58, 2000.

[Amb04] Ambiencia Information Systems, Inc. Galaxy application environment. http://www.ambiencia.com, 2004. Breckenridge.

[AMC+05] V. Alves, P. Matos, L. Cole, P. Borba, and G. Ramalho. Extracting and evolving mobile games product lines. In J.H. Obbink and K. Pohl, editors, *Proc. of the 9th International Software Product Line Conference (SPLC'05)*, volume 3714 of *Lecture Notes in Computer Science*, pages 70–81. Springer-Verlag Berlin, 2005. twiki.cin.ufpe.br/twiki/pub/SPG/AspectProductLine/AlvesSPLC05.pdf.

[AMC+07] V. Alves, P. Matos, L. Cole, A. Vasconcelos, P. Borba, and G. Ramalh. Extracting and evolving code in product lines with aspect-oriented programming. In *Transactions on Aspect-Oriented Software Development (TAOSD): Special Issue on Software Evolution*, 2007.

[APB+99] M. Abrams, C. Phanouriou, A. Batongbacal, S. Williams, and J. Shuster. UIML: An Appliance-Independent XML User Interface Language. In *Proc. of 8th World-Wide Web Conference WWW'8*, volume 31, pages 1695–1708, Amsterdam, Netherlands, 1999.

[APSA01] F.M. Ali, M.A. Perez-Quiñones, E. Shell, and M. Abrams. Building Multi-Platform User Interfaces with UIML. *ArXiv Computer Science e-prints*, 2001. Provided by the Smithsonian NASA Astrophysics Data System.

[Bal04] L. Balme. Infrastructure Logicielle pour Interfaces Homme-Machine Plastiques. In *Proc. of Secondes Rencontres Jeunes Chercheurs en Interaction Homme-Machine (RJC-IHM'04)*, pages 27–32, 2004.

[Bar97] J. E. Bardram. I love the system. I just don't use it! In *Proc. of the International ACM SIGGROUP Conference on Supporting Group Work (GROUP'97)*, pages 251–260, New York, NY, USA, 1997. ACM Press. ISBN: 0-89791-897-5.

[Bar02] N. Barralon. Interfaces Homme-Machine de Transition. Technical report, Ecole Doctorale de Informatique et Mathématiques Appliquées, Institut National Polytechnique de Grenoble, Université Joseph Fourier, Grenoble, 2002.

[Bar03] S. J. Barnes. Location-Based Services: The State of the Art. *e-Service Journal*, 2(3):59–70, 2003.

[BBPP04] P. Brézillon, M. Borges, J. Pino, and J.C. Pomerol. Context-awareness in group work: three case studies. In R. Meredith, G. Shanks, D. Arnott, and S. Carlsson, editors, *Proc. of IFIP International Conference on Decision Support Systems Decision Support in Uncertain and Complex World*, pages 115–124. Monash University, Australia (CD Rom), 7 2004. ISBN: 0 7326 2269 7.

[BC98] P. de Bra and L. Calvi. Aha an open adaptive hypermedia architecture. *The New Review of Hypermedia and Multimedia*, 4:115–140, 1998. Taylor Graham Publishers.

[BC99] S.A. Brewster and P.G. Cryer. Maximising screen-space on mobile computing devices. In *First published in CHI '99, Human Factors in Computing Systems: the CHI is the Limit*, volume 2 of *Late breaking results: overcoming human limitations*, pages 224–225. ACM Press Pittsburgh, Pennsylvania, USA, 1999. ISBN: 1-58113-158-5.

[BC01] H.E. Byun and K. Cheverst. Exploiting User Models and Context-Awareness to Support Personal Daily Activities. In *Proc. of Workshop on User Modelling for Context-Aware Applications (User Modelling Conference)*, 2001.

[BCC00] F. Bérard, J. Coutaz, and J. Crowley. Le Tableau Magique: Un Outil pour l'Activité de Reflexion. In *Proc. of the Conférence ErgoIHM'00 - CRT ILS ESTIA*, pages 33–40. D.L. Scapin and E. Vergisson, 2000.

[BCR05] A. Boronat, J.A. Carsí, and I. Ramos. Automatic Reengineering in MDA Using Rewriting Logic as Transformation Engine. In *Proceedings of the 9th European Conference on Software Maintenance and Reengineering (CSMR'05)*, pages 228–231. IEEE Computer Society Washington, 2005. ISBN: 0-7695-2304-8.

[BDB+04] L. Balme, A. Demeure, N. Barralon, J. Coutaz, and G. Calvary. CAMELEON-RT: A Software Architecture Reference Model for Distributed, Migratable and Plastic User Interfaces. In *Proc. of EUSAI*, pages 291–302, 2004.

[Bei00] M. Beigl. Memoclip: A location-based remembrance appliance. *Personal Ubiquitous Computing*, 4(4):230–233, 2000.

[Ber94] N.O. Bernsen. Foundations of Multimodal Representations: A Taxonomy of Representational Modalities. *Interacting with Computers*, 6(4):347–371, 1994.

[BFJ+01] G. Buchanan, S. Farrant, M. Jones, H. Thimbleby, Marsden G., and M. Pazzani. Improving mobile internet usability. In *Proc. of the 10th international conference on World Wide Web (WWW'01)*, pages 673–680, New York, NY, USA, 2001. ACM Press. ISBN: 1-58113-348-0.

[BFO02] C. Bey, E. Freeman, and J. Ostrem. Palm Os Programmer's Companion. Palm Inc. Volume 1, 2002.

[BGB03] S. Banerjee, A. Gupta, and A. Basu. Online Transcoding of Web Pages for Mobile Devices. In *Proc. of Mobile HCI'03*, pages 271–285. Springer Verlag, 2003.

[BHKN96] H. Balzert, F. Hofmann, V. Kruschinski, and C.Ñiemann. The JANUS Application Development Environment - Generating More than the User Interface. In Jean Vanderdonckt, editor, *Proc. of the 2nd International Workshop on Computer-Aided Design of User Interfaces: (CADUI'96) Computer-Aided Design of User Interfaces I*, pages 183–208. Presses Universitaires de Namur, 1996. ISBN: 2-87037-232-9.

[BHL+95] F. Bodart, A-M. Hennebert, J-M. Leheureux, I. Provot, B. Sacr, and J. Vanderdonckt. Towards a Systematic Building of Software Architectures: the TRIDENT Methodological Guide. In *Proc. of DSV-IS*, pages 262–278, 1995.

[BHLV94] F. Bodart, A.M. Hennebert, J.M. Leheureux, and J. Vanderdonckt. Towards a dynamic strategy for computer-aided visual placement. In *Proc. of 2nd ACM Workshop on Advanced Visual Interfaces (AVI'94)*, pages 78–87, Bari, Italy, 1994. ACM Press, New York, NY, USA. ISBN: 0-89791-733-2.

[Bir04] D. Birov. Aspects pattern oriented architecture for distributed adaptive mobile applications. In *Proc. of the 5th international conference on Computer systems and technologies (CompSysTech'04)*, pages 1–6, New York, NY, USA, 2004. ACM Press. ISBN: 954-9641-38-4.

[BJR99] G. Booch, I. Jacobson, and J. Rumbaugh. *El Lenguaje Unificado de Modelado.* Addison Wesley Longman. ISBN 8478290281, 1999.

[BMR+96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture: A System of Patterns*, volume 1. John Wiley & Sons, Inc., New York, NY, USA, 1st. edition, August 1996.

[BNR90] D. Browne, M. Norman, and D. Riches. Why Build Adaptive Interfaces. In D. Browne, P. Totterdell, and M. Norman, editors, *Adaptive User Interfaces*, pages 15–57. Academic Press, London, 1990.

[Boa95] M. Boasson. The artistry of software architecture. *IEEE Software*, 12(6):13–16, 1995.

[Bou06] Laurent Bouillon. *Reverse Engineering of Declarative User Interfaces*. PhD thesis, Université Catholique de Louvain, Louvain-la-Neuve, Belgium, 2006.

[BP90] R. Bastide and P. Palanque. Petri nets with objects for the design, validation and prototyping of userdriven interfaces. In D. Diaper, D. Gilmore, G. Cockton, and B. Shackel, editors, *Proc. of INTERACT'90: 3rd IFIP International Conference on Human-Computer Interaction*, pages 625–638, Amsterdam, The Netherlands, 1990. Elsevier.

[BP99a] M.R. Borges and J.A. Pino. Awareness mechanisms for coordination in asynchronous CSCW. In *Proc. of the WITS'99 Conference*, volume 1, pages 69–74, 1999.

[BP99b] P. Brezillon and J. Pomerol. Contextual knowledge sharing and cooperation in intelligent assistant systems. Le Travail Humain 62 (3), PUF, Paris, 1999. 223–246.

[BP04] R. Bandelloni and F. Paternò. Flexible Interface Migration. In *Proc. of the 9th International Conference on Intelligent User Interfaces (IUI'04)*, pages 148–155, Funchal, Madeira, Portugal, 2004. ACM press. New York, NY, USA. ISBN: 1-58113-815-6.

[BP05] S. Berti and F. Paternò. Migratory Multimodal Interfaces in Multidevice Environments. In *Proc. of the 7th International Conference on Multimodal Interfaces (ICMI'05)*, pages 92–99, Torento, Italy, 2005. ACM Press. ISBN: 1-59593-028-0.

[Bro04] A. Brown. An Introduction to Model Driven Architecture. Part I: MDA and Today's Systems. Systems IBM developerWorks, 2004. http://www-128.ibm.com/developerworks/rational/library/3100.html.

[Bru96] P. Brusilovsky. Methods and Techniques of Adaptive Hypermedia. *User Modeling and User-Adapted Interaction*, 6(2-3):87–129, 1996. Alfred Kobsa - Ten Year Aniversari.

[Bru99] P. Brusilovsky. Adaptive and intelligent technologies for web-based education. *Kunstliche Intelligenz (KI), Special Issue on Intelligent Systems and Teleteaching*, 13(4):19–25, 1999.

[BS97] T.W. Bickmore and B.N. Schilit. Digestor: device-independent access to the World Wide Web. *Computer Networks and ISDN Systems*, 29(8–13):1075–1082, 1997.

[BSW96] P. Brusilovsky, E. Schwarz, , and G. Weber. A tool for developing adaptive electronic textbooks on WWW. In *In Proc. of WebNet'96 of the AACE - World Conference of the Web Society*, pages 64–69. AACE, 1996. http://www.contrib.andrew.cmu.edu/ plb/WebNet96.html.

[Bur07] M. Burgués. Contenidos de última generación. Monográfico especial telefonía 3G. *La Vanguardia*, 2007. 10 de Febrero.

[BV94] F. Bodart and J. Vanderdonckt. On the problem of selecting interaction objects. In G. Cockton, S.W. Draper, and G.R.S. Weir, editors, *Proc. of BCS Conference HCI'94 (People and Computers IX)*, pages 163–178, New York, NY, USA, 1994. Cambridge University Press. ISBN: 0-521-48557-6.

[BVC04] L. Bouillon, J. Vanderdonckt, and K.C. Chow. Flexible re-engineering of web sites. In *Proc. of the International Conference on Intelligent User Interfaces (IUI'04)*, pages 132–139, New York, NY, USA, 2004. ACM Press. ISBN: 1-58113-815-6.

[BVS02] L Bouillon, J. Vanderdonckt, and N Souchon. Recovering Alternative Presentation Models of a Web Page with VAQUITA. In *Proc. of 4th Int. CADUI*, pages 311–322, Dordrecht, 2002. Kluwer academics. capítulo 27.

[BW98] A.W. Brown and K.C. Wallnau. The current state of CBSE. *IEEE Software*, 15(5):37–46, 1998.

[Cab07] A. Cabanillas. Glosario sobre telefonía 3G. Monográfico especial telefonía 3G. *La Vanguardia*, 2007. 10 de Febrero.

[Cac03] C. Cachero. *Una extensión a los métodos OO para el modelado y generación automática de interfaces hipermediales (OO-H'03)*. PhD thesis, University of Alicante (Spain), 2003.

[Cal98] G. Calvary. *Proactivité et Réactivité: de l'Assignation à la Complémentarité en Conception et Evaluation d'Interfaces Homme-Machine*. PhD thesis, Université Joseph-Fourier-Grenoble I, France, 1998.

[CAM04] CAMELEON Project. Plasticity of user interfaces. http://giove.cnuce.cnr.it/cameleon.htm, 2004.

[CC03] A.T.S. Chan and S.N. Chuang. Mobipads: A reflective middleware for context-aware mobile computing. *IEEE Transactions on Software Engineering*, 29(12):1072–1085, 2003. http://www.informatik.uni-trier.de/~ley/db/journals/tse/tse29.html#ChanC03.

[CCB00] J. Crowley, J. Coutaz, and F. Bérard. Things that see. In *Communications of the ACM*, volume 2498, pages 54–64, 2000.

[CCD+04] G. Calvary, J. Coutaz, O. Dâassi, L. Balme, and A. Demeure. Towards a new generation of widgets for supporting software plasticity: the 'Comet'. In *Proc. of Engineering Human Computer Interaction and Interactive Systems (EHCI-DVIS 2004)*, volume 3425 of *LNCS*, pages 306–324, 2004. ISBN 3-540-26097-8.

[CCL00] W. Cazzola, S. Chiba, and T. Ledoux. Reflection and meta-level architectures: State of the art and future trends. In J. Malenfant, S. Moisan, and A.M.D. Moreira, editors, *Proc. of the Workshops, Panels, and Posters on Object-Oriented Technology (ECOOP'00)*, volume 1964 of *Lecture Notes in Computer Science*, pages 1–15, London, UK, 2000. Springer-Verlag. ISBN: 3-540-41513-0.

[CCRR02] J. Crowley, J. Coutaz, G. Rey, and O. Reignier. Perceptual components for context-aware computing. In *Proc. of International Conference on Ubiquitous Computing (UbiComp'2002)*, volume 2498 of *LCNS*, pages 117–134, Göteborg, 2002. Springer-Verlag.

[CCT00] G. Calvary, J. Coutaz, and D. Thevenin. Embedding plasticity in the development process of interactive systems. In *Proc. of the Handheld and Ubiquitous Computing (HUC) Conference*, 2000.

[CCT01a] G. Calvary, J. Coutaz, and D. Thevenin. Supporting Context Changes for Plastic User Interfaces: A Process and a Mechanism. In A. Blandford, J. Vanderdonckt, and Ph. Gray, editors, *Proc. of Human-Computer Interaction (IHM-HCI)*, volume 1, pages 349–363, Lille, 2001. Springer-Verlag, London.

[CCT01b] G. Calvary, J. Coutaz, and D. Thevenin. A unifying reference framework for the development of plastic user interfaces. In *Proc. of the 8th IFIP International Conference on Engineering for Human-Computer Interaction (EHCI'01)*, pages 173–192, London, UK, 2001. Springer-Verlag.

[CCT+02a] G. Calvary, J. Coutaz, D. Thevenin, L. Bouillon, M. Florins, Q. Limbourg, N. Souchon, J. Vanderdonckt, L. Marucci, F. Paternò, and C. Santoro. The CAMELEON Reference Framework, Deliverable D1.1. Technical report, CAMELEON Project 2002, 2002.

[CCT+02b] G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, N. Souchon, L. Bouillon, M. Florins, and J. Vanderdonckt. Plasticity of User Interfaces: A Revised Reference Framework. In *Proc. of the First International Workshop on Task Models and Diagrams for User Interface Design (TAMODIA'02)*, pages 127–134, 2002.

[CCT+03] G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, and J. Vanderdonckt. A Unifying Reference Framework for Multi-target User Interfaces. *Journal of Interacting with Computers*, 15(3):289–308, 2003.

[CDME01] K. Cheverst, N. Davies, K. Mitchell, and C. Efstratiou. Using Context as a Crystal Ball: Rewards and Pitfalls. *Personal Ubiquitous Computing*, 5(1):8–11, 2001.

[CE00] K. Czarnecki and U.W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley Professional. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1st. edition, 2000.

[CEM03] L. Capra, W. Emmerich, and C. Mascolo. Carisma: Context-aware reflective middleware system for mobile applications. *IEEE Transactions on Software Engineering*, 29(10):929–945, 2003. http://www.cs.ucl.ac.uk/staff/L.Capra/research/carisma.html.

[CFJ03] H. Chen, T. Finin, and A. Joshi. Using owl in a pervasive computing broker. In S. Cranefield, T.W. Finin, V.A.M. Tamma, and S. Willmott, editors, *Proc. of the Workshop on Ontologies in Agent Systems (OAS 2003) at the 2nd International Joint Conference on Autonomous Agents and Multi-Agent Systems*, volume 73 of *CEUR Workshop Proc.*, pages 9–16. CEUR-WS.org, 2003. http://CEUR-WS.org/Vol-73/.

[CGPO02] C.A. Collazos, L.A. Guerrero, J.A. Pino, and S.F. Ochoa. Introducing knowledge-shared awareness. In M. Boumedine, editor, *Proc. of IASTED International Conference: Information and Knowledge Sharing (IKS'02)*, volume 361-061, pages 13–18, Anaheim, Calgary, Zurich, November 2002. ACTA Press. ISBN: 0-88986-325-3.

[Che76] P.P. Chen. The Entity-Relationship Model - Towards a Unified View of Data. *ACM Transaction Database System*, 1(1):9–36, 1976.

[CK00] G. Chen and D. Kotz. A Survey of Context-Aware Mobile Computing Research. Technical Report TR2000–381, Hanover, NH, USA, 2000.

[CL99] L.L. Constantine and L.A.D. Lockwood. *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*. Addison-Wesley Professional, 1st edition, 1999.

[CLC04a] T. Clerckx, K. Luyten, and K. Coninx. Dynamo-AID: A Design Process and a Runtime Architecture for Dynamic Model-Based User Interface Development. In *Proc. of EHCI DSV-IS'04*, volume 3425 of *LNCS*, pages 77–95, 2004.

[CLC04b] T. Clerckx, K. Luyten, and K. Coninx. Generating Context-Sensitive Multiple Device Interfaces from Design. In *Proc. of the 4th International Conference on Computer-Aided Design of User Interfaces (CADUI'04)*, pages 288–301, 2004.

[CLV+03] K. Coninx, K. Luyten, C. Vandervelpen, J. Van den Bergh, and B. Creemers. Dygimes: Dynamically Generating Interfaces for Mobile Computing Devices and Embedded Systems. In *Proc. of Mobile HCI'03*, pages 256–270. Springer-Verlag, 2003.

[CM03] C. Correa and I. Marsic. A Flexible Architecture to Support Awareness in Heterogeneous Collaborative Environments. In *Proc. of CTS*, pages 69–77, 2003.

[CMN83] S.K. Card, T.P. Moran, and A. Newell. *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates, Inc., Mahwah, NJ, USA, 1983.

[CMP04] F. Correani, G. Mori, and F. Paternò. Supporting Flexible Development of Multi-Device Interfaces. In *Proc. of EHCI/DS-VIS*, pages 346–362, Hamburg, 2004. Springer-Verlag.

[CN01] P Clements and L.M Northrop. *Software product lines: practices and patterns*. The SEI Series in Software Engineering. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[CNY96] L. Chung, B.A. Nixon, and E.S.K. Yu. Dealing with change: An approach using non-functional requirements. *Requirements Engineering*, 1(4):238–260, 1996. Springer-Verlag.

[Col05] A. Colyer. AOP@Work: Introducing AspectJ 5. A first look at java 5 support in AspectJ, and other new features. IBM Developerworks, IBM, 2005.

[Con03] L.L. Constantine. Canonical Abstract Prototypes for Abstract Visual an Interaction Design. In J.A. Jorge, N.J. Nunes, and J.F. Cunha, editors, *Design, Specification and Verification of Interactive Systems (DSV-IS'03)*, volume 2844/2003 of *Lecture Notes in Computer Science*, page 304. Springer Verlag, 2003. ISBN: 978-3-540-20159-5.

[Cou98] J. Coutaz. Interfaces Homme-Machine: le futur ne manque pas d'avenir. In *Proc. of Ergonomie et Informatique Avancée*, pages 43–55, Biarritz, France, 1998. ISBN: 2-9503134-9-3.

[Cov01] R. Cover. XML Markup Languages for User Interface Definition. http://xml.coverpages.org/userInterfaceXML.html, 2001.

[CPA04] W. Cazzola, S. Pini, and M. Ancona. Evolving pointcut definition to get software evolution. In W. Cazzola, S. Chiba, and G. Saake, editors, *Proc. of European Conference on Object-Oriented Programming (RAM-SE'04-ECOOP'04) Workshop on Reflection, AOP, and Meta-Data for Software Evolution*, pages 83–88. Fakultät für Informatik, Universität Magdeburg, 2004.

[CPR99] R.M. Carro, E. Pulido, and P. Rodríguez. Designing adaptive web-based courses with tangow. In G. Cumming, T. Okamoto, and L. Gómez, editors, *Proc. of the 7th International Conference on Computers in Education, (ICCE'99)*, volume 2, pages 697–704. Amsterdam: IOS Press, 1999.

[CR02] J. Coutaz and G. Rey. Foundations for a Theory of Contextors. In K. Kolski and J. Vanderdonckt, editors, *Proc. of the 4th International Conference on Computer-Aided Design of User Interfaces, (CADUI'02)*, pages 283–303. Kluwer Academic Publications, 2002. ISBN: 1-4020-0643-8.

[CS89] H. Clarck and E. Schaefer. Contributing to discourse. *Cognitive Science*, 13(2):259–294, 1989.

[DA00a] A.K. Dey and G.D. Abowd. Cyberminder: A context-aware system for supporting reminders. In *Proc. of the 2nd International Symposium on Handheld and Ubiquitous Computing ((HUC2K))*, volume 1, page 172, Bristol, UK, 2000. Springer Verlag. http://www.cc.gatech.edu/fce/contexttoolkit/pubs/HUC2000.pdf.

[DA00b] A.K. Dey and G.D. Abowd. Towards a Better Understanding of Context and Context-Awareness. In *Proc. of the Workshop on the What, Who, Where, When, Why and How of Context-Awareness (CHI'00)*, The Hague, The Netherlands, 2000.

[DAS01] A. Dey, G. Abowd, and D. Salber. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human Computer Interaction Journal*, 16(2–4):97–166, 2001. Special Issue on Context-Aware Computing.

[DB92] P. Dourish and V. Belloti. Awareness and coordination in shared workspaces. In *Proc. of the ACM conference on Computer-supported cooperative work (CSCW'92)*, pages 107–114, New York, NY, USA, 1992. ACM Press. ISBN: 0-89791-542-9.

[DB03] A. Dantas and P. Borba. Developing adaptive j2me applications using AspectJ. *j-jucs : Journal of Universal Computer Science*, 9(8):935–955, 2003.

[DC03] A. Demeure and G. Calvary. Plasticity of user interfaces: towards an evolution model based on conceptual graphs. In *Proc. of the 15th French-speaking Conference on Human-Computer Interaction on 15eme Conference Francophone sur l'Interaction Homme-Machine (IHM'03)*, volume 51, pages 80–87. ACM Press, 2003. ISBN 1-58113-803-2.

[DCCD03] O. Dâassi, G. Calvary, J. Coutaz, and A. Demeure. Comet : Une nouvelle génération de 'widget' pour la Plasticité des Interfaces. In *Proc. of IHM 2003*, pages 64–71. ACM Press, 2003.

[Del04] T. Delclós. España ante el reto de la Sociedad de la Información. *El país, Suplemento Sociedad*, 18 de Octubre 2004.

[Dey98] A.K. Dey. Context-aware computing: The cyberdesk project. In *Proc. of Spring Symposium on Intelligent Environments (AAAI'98)*, pages 51–54, Stanford University, 1998. http://www.cc.gatech.edu/fce/cyberdesk/pubs/AAAI98/AAAI98.html.

[Dey00] A.K. Dey. *Providing architectural support for building context-aware applications.* Thesis dissertation, College of Computing, Georgia Institute of Technology, 2000. ISBN: 0-493-01246-X; Director:Gregory D. Abowd.

[DFAB03] A. Dix, J. Finlay, G. Abowd, and R. Beale. *Human-Computer Interaction.* Pearson Education, 3rd edition, 2003.

[DFS02] R. Duoence, P. Fradet, and M. Südholt. Detection and resolution of aspect interactions. Technical Report 4435, Institut National de Recherche en Informatiqueet en Automatique INRIA, France, 2002.

[DL89] J. Dowell and J. Long. *Towards a conception for an engineering discipline of human factors*, volume 32, pages 1513–1535. 1989.

[DLH02] D. van Duyne, J. Landay, and J. Hong. *The Design of Sites: Patterns, Principles and Proceses for Crafting a Customer-Centered Web.* Addison-Wesley Pub. Co., 2002.

[DMCB98] N. Davies, K. Mitchell, K. Cheverst, and G. Blair. Developing a context sensitive tourist guid. Technical report computing department, Lancaster University, Glasgow, 1998.

[DMKS93] H. Dieterich, U. Malinowski, T. Kühme, and M. Schneider-Hufschmidt. State of the art in adaptive user interfaces. In M. Schneider-Hufschmidt, T. Khüme, and U. Malinowski, editors, *Adaptive User Interfaces: Principles and Practice, Human Factors in Information Technology, 10*, chapter Part 1: Setting the Stage, pages 13–48. North Holland: Elsevier Science Publishers B.V., 1993. ISBN 10:0-444-81545-7, 13:978-0-444-81545-3.

[dRE05] R.C.A. da Rocha and M. Endler. Evolutionary and efficient context management in heterogeneous environments. In *Proc. of the 3rd international workshop on Middleware for pervasive and ad-hoc computing (MPAC'05)*, volume 115, pages 1–7, New York, NY, USA, 2005. ACM Press. ISBN: 1-59593-268-2.

[DYBJ04] A. Dantas, J.W. Yoder, P. Borba, and R. Johnson. Using aspects to make adaptive object-models adaptable. *European Conference on Object Oriented Programming (RAM-SE'04-ECOOP'04)*, pages 9–19, 2004. Workshop Position Paper.

[EGR91] C.A. Ellis, S.J. Gibbs, and G.L. Rein. Groupware: some issues and experiences. *Communications of the ACM*, 34(1):39–58, 1991. ACM Press.

[Ens02] J. Ensing. Software architecture for the support of context aware applications. Preliminary study, (Delft University of Technology). Koninklijke Philips Electronics N.V., 2002. http://www.extra.research.philips.com/publ/rep/nl-ur/NL-UR2002-841.pdf.

[EP00] J. Eisenstein and A. Puerta. Adaptation in automated user-interface design. In *Proc. of the 5th International Conference on Intelligent User Interfaces (IUI'00)*, pages 74–81, New Orleans, Louisiana, United States, 2000. ACM Press. ISBN: 1-58113-134-8.

[ES95] T. Elwert and T. Schlungbaum. Modelling and Generation of Graphical User Interfaces in the TADEUSApproachh. In *Designing, Specification and Verification of Interactive Systems*, pages 193–208, Vienna, 1995. Springer Editions.

[EVP00] J. Eisenstein, J. Vanderdonckt, and A. Puerta. Adapting to mobile contexts with user-interface modeling. In *Proc. of the Third IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'00)*, pages 83–92, Monterrey, 2000. IEEE Computer Society. ISBN: 0-7695-0816-2.

[EVP01] J. Eisenstein, J. Vanderdonckt, and A.R. Puerta. Applying model-based techniques to the development of uis for mobile computer. In *Proc. of the 6th international conference on Intelligent User Interfaces (IUI'01)*, pages 69–76, New York, NY, USA, 2001. ACM Press. ISBN: 1-58113-325-1.

[Fer05] A. Fernández. *Groupware for Collaborative Tailoring.* PhD thesis, FernUniversitä Hagent, Fachbereich Informatik, 2005.

[FF04] R.E. Filman and D.P. Friedman. Aspect-oriented programming is quantification and obliviousness. In R.E. Filman, T. Elrad, S. Clarke, and M. Aksit, editors, *Aspect-Oriented Software Development*, chapter 1.2 LANGUAGES AND FOUNDATIONS, pages 21–35. Addison-Wesley, Boston, 2004. ISBN10: 0-321-21976-7; 13:9780321219763.

[FHLS98] G. Froehlich, J. Hoover, L. Liu, and P. Sorenson. Designing object-oriented frameworks. In S. Zamir, editor, *CRC Handbook of Object Technology*, chapter SECTION IV Object-Oriented Frameworks, pages 25–1 – 25–22. CRC Press, Boca Raton, FL., 1998. http://www.cs.ualberta.ca/s̃ofteng/papers/design12.pdf.

[Fis01] G. Fisher. User Modeling in Human-Computer Interaction. *User Modeling and User-Adapted Interaction*, 11(1–2):65–86, 2001.

[FJ05] L. Fuentes and D. Jiménez. An aspect-oriented ambient intelligence middleware platform. In *Proc. of the 3rd international workshop on Middleware for pervasive and ad-hoc computing (MPAC'05), ACM International Conference*, volume 115, pages 1–8, New York, NY, USA, 2005. ACM Press. ISBN: 1-59593-268-2.

[FJP05] L. Fuentes, D. Jiménez, and M. Pinto. Development of ambient intelligence applications using components and aspects. In *Proc. of (UCAMI'05), Journal of Universal Computer Science*, volume 12, pages 236–251, 2005.

[FKKM91] J.D. Foley, W.C. Kim, S. Kovacevic, and K. Murray. UIDE, an Intelligent User Interface Design Environment. *Intelligent User Interfaces*, 18(20):339–384, 1991. Chapter 15, ISBN: 0-201-50305-0.

[FKL+98] S. Fussell, R. Kraut, F. Lerch, W. Scherlis, N. McNally, and J. Cadiz. Coordination, overload and team performance: effects of team communication strategies. In *Proc. of the 1998 ACM conference on Computer supported cooperative work (CSCW'98)*, pages 275–284, New York, NY, USA, 1998. ACM Press. ISBN: 1-58113-009-0.

[FKV00] D. Fritsch, D. Klinec, and S. Volz. Nexus positioning and data management concepts for location aware applications. In *Proc. of the 2nd International Symposium on Telegeoprocessing*, pages 171–184, 2000. http://www.nexus.uni-stuttgart.de/.

[Fow96] M Fowler. *Analysis Patterns: Reusable Object Model*. The Addison-Wesley Object Technology Series. Addison-Wesley Professional, 1st edition, 1996.

[FRPG04] J. Favela, M. Rodríguez, A. Preciado, and V. González. Integrating context-aware public displays into a mobile hospital information system. *IEEE Transactions on Information Technology in Biomedicine*, 8(3):279–286, 2004.

[FS97] M. Fayad and D.C. Schmidt. Object-oriented application frameworks. *Communications of the ACM*, 40(10):32–38, 1997.

[FSJ99] M.E. Fayad, D.C. Schmidt, and R.E. Johnson. *Building Application Frameworks: Object-Oriented Foundations of Framework Design*. John Wiley & Sons, New York, NY, USA, 1st. edition, 1999.

[FV04] M. Florins and J. Vanderdonckt. Graceful degradation of user interfaces as a design method for multiplatform systems. In *Proc. of the 9th international conference on Intelligent User Interfaces (IUI'04)*, pages 140–147, Funchal, Madeira, Portugal, 2004. ACM Press. New York, NY, USA. ISBN: 1-58113-815-6.

[FVB+01] E. Furtado, J. Vasco, W. Bezerra, D. William, L. da Silva, Q. Limbourg, and J. Vanderdonckt. An Ontology-Based Method for Universal Design of User Interfaces. In *Proc. of Workshop on Multiple User Interfaces over the Internet: Engineering and Applications Trends*, 2001.

[Gal07] N. Gallego. Apple irrumpe en la telefonía móvil con un nuevo modelo de iPod. *La Vanguardia. Monográfico especial Economía*, 10 de Enero 2007.

[GBM+99] T. Griffiths, P. Barclay, J. McKirdy, N. Paton, P. Gray, J. Kennedy, R. Cooper, C. Goble, A. West, and M. Smyth. Teallach: A Model-Based User Interface Development Environment for Object Databases. In Norman W.P. and T. Griffiths, editors, *Proc. of User Interfaces to Data Intensive Systems (UIDIS'99)*, pages 86–96. IEEE Computer Society Publishers, 1999. ISBN: 0-7695-0262-8.

[GCG06] M. P. González, C. A. Collazos, and T. Granollers. Guidelines and usability principles to design and test shared-knowledge awareness for a cscl interface. In Y.A. Dimitriadis, I. Zigurs, and E. Gómez-Sánchez, editors, *Proc. of Groupware: Design, Implementation, and Use, 12th International Workshop (CRIWG'06)*, volume 4154 of *Lecture Notes in Computer Science*, pages 102–117. Springer, 2006. ISBN: 3-540-39591-1.

[GCP01] J. Gómez, C. Cachero, and O. Pastor. Conceptual Modeling of Device-Independent Web Applications. *IEEE MultiMedia*, 8(2):26–39, 2001.

[GFP+98] T. Griffiths, G. Forrester, N. Paton, J. Kennedy, P. Barclay, R. Cooper, C. Goble, and P. Gray. Exploiting Model-based Techniques for User Interfaces to Databases. In Y. Ioannidids and W. Klass, editors, *Proc. of the IFIP TC2/WG. 4th Working Conference on Visual Database Systems 4*, pages 21–46, London, UK, UK, 1998. Chapman & Hall, Ltd. ISBN: 0-412-84400-1.

[GG02] C. Gutwin and S. Greenberg. A Descriptive Framework of Workspace Awareness for Real-Time Groupware. *Computer Supported Cooperative Work*, 11(3):411–44, 2002. Kluwer Academic Publishers. Norwell.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st. edition, 1995.

[GL03] J.D. Gradecki and N. Lesiecki. *Mastering AspectJ: Aspect-Oriented Programming in Java*. John Wiley and Sons, 2003.

[GMP+98] T. Griffiths, J. McKirdy, N. Paton, J. Kennedy, R. Cooper, P. Barclay, C. Goble, P. Gray, M. Smyth, and A. Dinn. An open model-based interface development system: The teallach approach. In P. Markopoulos and P. Johnson, editors, *Proc. of DSV-IS'98*, pages 32–49, Abingdon, 1998. Eurographics.

[Gon05a] A. González. Curiositats i noves aplicacions. Suplement especial Telefonia mòbil. *El Periódico*, 28 de Febrero 2005.

[Gon05b] A. González. No sense mòbil. Suplement especial Telefonia mòbil. *El Periódico*, 28 de Febrero 2005.

[GOPC06] L.A. Guerrero, S.F. Ochoa, J.A. Pino, and C.A. Collazos. Selecting computing devices to support mobile collaboration. In *Group Decision and Negotiation*, volume 15 of *Empresas y Economía*, pages 243–271. Springer Netherlands, 2006. ISSN: 0926-2644 (Impreso) 1572-9907 (On-line).

[Gru94] J. Grudin. Groupware and social dynamics: eight challenges for developers. *Communications of the ACM*, 37(1):92–105, 1994. ISBN: 1-55860-246-1.

[GRV01] D. Grolaux, P. van Roy, and J. Vanderdonckt. QTk: An integrated Model-Based aprroach to designing executable user interfaces. In Ch. Johnson, editor, *Proc. of 8th International Workshop on Design, Specification, Verification of Interactive Systems (DSV-IS'01)*, pages 77–91. Springer-Verlag, 2001. GIST Tech. Report G-2001-1, Dept. of Computer Science, University of Glasgow, Scotland.

[GWC+00] T.C.N. Graham, L. Watts, G. Calvary, J. Coutaz, E. Dubois, and L.Ñigay. A dimension space for the design of interactive systems within their physical environments. In *Proc. of the conference on Designing interactive systems (DIS'00)*, pages 406–416, New York City, New York, United States, 2000. ACM Press. ISBN: 1-58113-219-0.

[Har87] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[HBH+98] E. Horvitz, J. Breese, D. Hecherman, D. Hovel, and K. Rommelse. The Lumiere Project: Bayesian User Modelling for Inferring the Goals and Needs of Software Users. In *Proc. of the 14th Conference on Uncertainty in Artificial Intelligence*, pages 256–265, Madison, WI., 1998. Morgan Kaufmann.

[HC02] S. Hanenberg and P. Costanza. Connecting aspects in AspectJ: Strategies vs. Pattern. In *Proc. of the 1rst Workshop on Aspects, Components, and Patterns for Infrastructure Software (AOSD)*, 2002.

[HHN86] E. L. Hutchins, J. D. Hollan, and D. A. Norman. Direct manipulation interfaces. In D. A. Norman and S. W. Draper, editors, *User Centered System Design: New Perspectives on Human-Computer Interaction*, pages 87–124. Lawrence Erlbaum Associates, Inc., Hillsdale, NJ, 1986. ISBN: 0898597811.

[HK02] S. Hanenberg and G. Kiczales. Design pattern implementation in java and AspectJ. In *Proc. of the 17th ACM SIGPLAN conference on Object-oriented pro-*

gramming, systems, languages, and applications: (OOPSLA'02), pages 161–173, New York, NY, USA, 2002. ACM Press.

[HKK04] H. Harroud, M. Khedr, and A. Karmouch. Building policy-based context aware applications for mobile environments. In A. Karmouch, L. Korba, and E.R.M. Madeira, editors, *Proc. of 1st Int. workshop Mobility Aware Technologies and Applications (MATA'04)*, volume 3284 of *Lecture Notes in Computer Science, Context-Aware Applications and Networks*, pages 48–61. Springer Berlin, 2004. ISBN: 3-540-23423-3.

[HL95] W.L. Hürsch and C.V. Lopes. Separation of concerns. Technical Report NU-CCS-95-03, College of Computer Science, Northeastern University, 1995.

[HL01] J.I. Hong and J.A. Landay. An infrastructure approach to context-aware computing. *Human-Computer Interaction, University of California at Berkeley*, 16(2, 3 & 4):287–303, 2001. http://www.cs.cmu.edu/ jasonh/publications/context-essay-final.pdf.

[HLM99] T. Highley, M. Lack, and P. Myers. Aspect oriented programming: A critical analysis of a new programming paradigm. Technical Report (CS-99-29), University of Virginia, Department of Computer Science, Charlottesville, VA, USA, 1999.

[HO93] W. Harrison and H. Ossher. Subject-oriented programming: a critique of pure objects. In ACM SIGPLAN Notices, editor, *Proc. of the eighth annual conference on Object-oriented programming systems, languages, and applications (OOPSLA'93)*, volume 28, pages 411–428, New York, NY, USA, 1993. ACM Press. ISBN: 0-89791-587-9.

[Hol90] J.D. Hollan. User Modeling and User Interfaces: A Case for Domain Models, Task Models, and Tailorability. In *Proc. of 8th National Conference of Artificial Intelligence (AAAI'90)*, page 1137, Boston, Massachusetts, 1990. AAAI Press. The MIT Press. ISBN 0-262-51057-X.

[Hor99] E. Horvitz. Principles of mixed-initiative user interfaces. In *Proc. of the SIGCHI conference on Human factors in computing systems (CHI'99)*, pages 159–166, New York, NY, US, 1999. ACM Press. ISBN: 0-201-48559-1.

[HPSH00] K. Hinckley, J. Pierce, M. Sinclair, and E. Horvitz. Sensing techniques for mobile interaction. In *Proc. of the 13th annual ACM Symposium on User Interface Software and Technology (UIST'00), CHI Letters*, pages 91–100, New York, NY, USA, 2000. ACM Press. ISBN: 1-58113-212-3.

[HUS03] S. Hanenberg, R. Unland, and A. Schmidmeier. Aspectj idioms for aspect-oriented software construction. In *Proc. of the 8th European Conference on Pattern Languages of Programs (EuroPLoP) held in conjunction with the 5th International Conference on Aspect-Oriented Software Development (AOSD'06)*, 2003.

[IBM93] IBM Corporation. *Object-Oriented Interface Design: IBM Common User Access Guidelines*. Que Corporation, Carmel, IN, Indianapolis, IN, USA, 1993.

[IEE07] IEEE Computer Society. IEEE Transactions on Mobile Computing. http://www.computer.org/portal/site/ieeecs/index.jsp, 2007.

[IFI97] IFIP: International Federation for Information Processing WG2.7. Design Principles for Interactive Software. http://xml.coverpages.org/userInterfaceXML.html, 1997. Chapman & Hall, Ltd. London, UK, UK. ISBN: 0-412-72470-7. Gram, C. and Cockton, G. (Eds.).

[ISO88] ISO/IS 8807. Information Process Systems – Open Systems Interconection – LOTOS, a formal description based on temporal ordering of observational behaviour. http://www.iso.org/iso/en/ISOOnline.frontpage, 1988.

[ISO91] ISO/IEC 9126. Software Product Evaluation. Quality characteristics and guidelines for their use. http://www.cse.dcu.ie/essiscope/sm2/9126ref.html, 1991.

[ISO03a] ISO/IEC 25021. Software and Systems Engineering - Software product quality requirements and evaluation (SquaRE) - Measurement, 2003.

[ISO03b] ISO/IEC CD 25000.2. Software and Systems Engineering - Software product quality requirements and evaluation (SquaRE) - Guide to SquaRE, 2003.

[JF88] R.E. Johnson and B. Foote. Designing reusable classes. *Objet-Oriented Programming*, 1(2):22–35, 1988.

[JWMP93] P. Johnson, S. Wilson, P. Markopoulos, and J. Pycock. ADEPT: Advanced Design Environment for Prototyping with Task Models. In *Proc. of the SIGCHI conference on Human factors in computing systems (CHI'93)*, page 56, New York, NY, USA, 1993. ACM Press. ISBN: 0-89791-575-5.

[JWZ93] C. Janssen, A. Weisbecher, and J. Ziegler. Generating user interfaces from data models and dialogue net specifications. In *Proc. of the ACM Conference on Human Factors in Computing Systems (INTERCHI'93)*, pages 418–423, New York, NY, USA, 1993. ACM Press. ISBN: 0-89791-575-5.

[KdRB91] G.J. Kiczales, J. des Rivieres, and D.G. Bobrow. *The Art of the Metaobject Protocol*. The MIT Press, Cambridge, MA, 1991.

[KHH+01a] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. Griswold. Getting started with AspectJ. *Communications of the ACM*, 44(10):59–65, 2001. ACM Press.

[KHH+01b] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W.G. Griswold. An overview of AspectJ. In *Proc. of the 15th European Conference on Object-Oriented Programming (ECOOP'01)*, volume 2072, pages 327–355, London, UK, 2001. Springer-Verlag. ISBN: 3-540-42206-4.

[Kic96] G. Kiczales. Aspect-oriented programming. *ACM Computing Surveys (CSUR)*, 28(4es.), 1996. Article No. 154.

[KKMZ03] A. Knapp, N. Koch, F. Moser, and G. Zhang. ArgoUWE: A case tool for Web applications. In J. Ralyté and C. Roland, editors, *Proc. of the 1rst International Workshop on Engineering Methods to Support Information Systems Evolution (EMSISE´03)*, pages 37–50, Genève, 2003.

[KLM+97] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. *Aspect-Oriented Programming*, volume 1241 of *Lecture Notes in Computer Science*, chapter Object-Oriented Programming: 11th European Conference (Ecoop'97), pages 220–242. Springer-Verlag, 1997. url: www.parc.com/research/projects/aspectj/downloads/ECOOP1997-AOP.pdf - ISBN: 3-540-63089-9.

[Koc00] N. Koch. *Software Engineering for Adaptive Hypermedia Systems: Reference Model, Modelling Techniques and Development Process*. PhD thesis, Ludwig-Maximilians-Universität Munchen, 2000. Reihe Softwaretechnik of FAST, Vol. 12, Uni-Druck Verlag.

[Kos90] T. Koschmann. *The Common Lisp companion*. John Wiley & Sons, Inc., New York, NY, USA, 1st. edition, 1990.

[KPRS01] G. Kappel, B. Pröll, W. Retschitzegger, and W. Schwinger. Modelling ubiquitous web applications - the WUML approach. In H. Arisawa, Y. Kambayashi, V. Kumar, H. C. Mayr, and I. Hunt, editors, *Revised Papers from the HUMACS, DASWIS, ECOMO, and DAMA on ER 2001 Workshops*, volume 2465 of *Lecture Notes In Computer Science*, pages 183–197, London, UK, 2001. Springer-Verlag. ISBN: 3-540-44122-0.

[KPRS03] G. Kappel, B. Pröll, W. Retschitzegger, and W. Schwinger. Customisation for ubiquitous web applications, a comparison of approaches 1. *International Journal of Web Engineering and Techonology (IJWET)*, 1(1):79–111, 2003. Business Informatics Group (BIG), Vienna University of Technology.

[KRK+02] G. Kappel, W. Retschitzegger, E. Kimmerstorfer, B. Pröll, W. Schwinger, and T. Hofer. Towards a Generic Customisation Model for Ubiquitous Web Applications. In *Proc. of the 2nd International Workshop on Web Oriented Software Technology (IWWOST) in conjunction with the 16th European Conference on Object-Oriented programming (ECOOP'02)*, pages 79–104, 2002. ISBN: 84-931538-9-3.

[Kru01] C.W. Krueger. Easing the transition to software mass customization. In *Revised Papers from the 4th International Workshop on Software Product-Family Engineering (PFE'01)*, volume 2290 of *Lecture Notes In Computer Science*, pages 282–293, London, UK, 2001. Springer-Verlag. ISBN: 3-540-43659-6.

[KW89] A. Kobsa and W. Wahlster. *User models in dialog systems.* Springer Series On Symbolic Computation And Artificial Intelligence. Springer-Verlag New York, Inc., New York, NY, USA, 1989.

[LAC05] M. Lorente, F. Asteasuain, and B.E. Contreras. Programemos en AspectJ. http://www.programemos.com/trabajos/programemos Version 1.1.

[Lad02] R. Laddad. I want my aop series. JavaWorld. Part I., 2002.

[Lad03] R. Laddad. *AspectJ in action. Practical Aspect-Oriented Programming.* Manning Publications Coorporatiton, Greenwich, CT, USA, 2003.

[Lad05] R. Laddad. Aop and metadata: A perfect match, part 1: Concepts and constructs of metadata-fortified aop and part 2: Multidimensional interfaces with metadata. IBM DeveloperWorks, 2005.

[Lal01] V. Lally. Analyzing teaching and learning interactions in a networked collaborative learning environment: issues and work in progress. In *Proc. of Euro CSCL'01*, 2001.

[Lar03] Craig Larman. *UML y patrones. Introducción al análisis y diseño orientado a objetos.* Perason educación S.A. Prentice Hall, 2nd. edition, 2003.

[LC01] K. Luyten and K. Coninx. An XML-Based Runtime User Interface Description Language for Mobile Computing Devices. In *Proc. of the 8th International Work-*

shop on Interactive Systems: Design, Specification, and Verification (DSV-IS'01), volume 2220 of *LNCS*, pages 17–29, 2001.

[LCCV03] K. Luyten, T. Clerckx, K. Coninx, and J. Vanderdonckt. Derivation of a Dialog Model from a Task Model by Activity Chain Extraction. In J.A. Jorge, N.J. Nunes, and J.F. Cunha, editors, *Proc. of the 10th International Workshop on Interactive Systems: Design, Specification, and Verification (DSV-IS'03)*, volume 2844 of *LNCC*, pages 203–217, 2003. ISBN: 3-540-20159-9.

[LF94] M. Lamming and M. Flynn. Forget-me-not. intimate computing in support of human memory. In *Proc. of FRIEND21'94 Symposium on Next Generation Human Interfaces*, Rank Xerox Research Centre, 1994. Also available as RXRC TR 94-103.

[LGR00] M.D. Lozano, P. González, and I. Ramos. User Interface Specification and Modeling in an Object Oriented Environment for Automatic Software Development. In Q. Li, D. Firesmith, R. Riehle, and B. Meyer, editors, *Proc. of IEEE 34th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS 34)*, pages 373–381, Santa Barbara, CA, USA, 2000. IEEE Computer Society. ISBN: 0-7695-0774-3.

[Lie95] K. Lieberherr. Workshop on adaptable and adaptive software. In *Addendum to the proc. of the 10th annual conference on Object-oriented programming systems, languages, and applications (Addendum) (OOPSLA'95)*, pages 149–154, New York, NY, USA, 1995. ACM Press. ISBN10: 0-89791-721-9; ISBN13: 978-0-89791-721-6.

[Lim04] Q. Limbourg. *Multi-Path Development of User Interfaces.* Ph. d. thesis, Université Catholique de Louvain, Institut d'Administration et de Gestion (IAG), Louvain-la-Neuve, Belgium, 2004.

[LKRF99] A. Leonhardi, U. Kubach, K. Rothermel, and A. Fritz. Virtual information towers-a metaphor for intuitive, location-aware information access in a mobile environment. In *Proc. of the 3rd IEEE International Symposium on Wearable Computers: (ISWC'99)*, pages 15–20, Washington, DC, USA, 1999. IEEE Computer Society. ISBN: 0-7695-0428-0.

[LL02a] T. Lemlouma and N. Layaïda. Universal profiling schemata (ups). http://opera.inrialpes.fr/people/Tayeb.Lemlouma/NegotiationSchema/, 2002.

[LL02b] T. Lemlouma and N. Layaïda. Device Independent Principles for Adapted Content Delivery. Technical report, INRIA, 2002.

[LL03a] T. Lemlouma and N. Layaïda. Adapted Content Delivery for Different Contexts. In *Proc. of the 2003 Symposium on Applications and the Internet (SAINT'03)*, pages 190–199, Washington, DC, USA, 2003. IEEE Computer Society. ISBN: 0-7695-1872-9.

[LL03b] T. Lemlouma and N. Layaïda. Smil Content Adaptation for Embedded Devices. In *Proc. of the Synchronised Multimedia Integration Language European Conference (SMIL'03)*, Paris, 2003.

[LL04] T. Lemlouma and N. Layaïda. Context-Aware Adaptation for Mobile Devices. In *Proc. of the 5th IEEE International Conference on Mobile Data Management (MDM'04)*, pages 106–111, Berkeley, CA, USA, 2004. IEEE Computer Society. ISBN: 0-7695-2070-7.

[LLCR03] K. Luyten, T. Van Laerhoven, K. Coninx, and F. Van Reeth. Runtime transformations for modal independent user interface migration. *Interacting with Computers*, 15(3):329–347, 2003. Elsevier Science.

[LMFL03] V. López-Jaquero, F. Montero, A. Fernández, and M. Lozano. Towards adaptive user interface generation: One step closer. In *5th International Conference on Enterprise Information Systems, (ICEIS'03)*, pages 97–103, 2003.

[LMM+03] V. López-Jaquero, F. Montero, J.P. Molina, A. Fernández-Caballero, and P. González. Model-Based Design of Adaptive User Interfaces through Connectors. In J.A. Jorge, N.J. Nunes, and J.F. e Cunha, editors, *Design, Specification and Verification of Interactive Systems 2003 (DSV-IS 2003)*, volume 2844 of *Lecture Notes in Computer Science*, pages 245–257, Funchal, Madeira Island, Portugal, 2003. Springer. ISBN: 3-540-20159-9.

[LO05] N. Layaïda and J. Von Ossenbruggen. SMIL 2.0 language profile. W3c recommendation, world wide web consortium, W3C, 2005. http://www.w3.org/TR/2005/REC-SMIL2-20050107/smil20-profile.html.

[Lóp05] V. López-Jaquero. *Interfaces de Usuario Adaptativas Basadas en Modelos y Agentes de Software*. PhD thesis, Universidad de Castilla-La Mancha, 2005.

[LRSP98] P. Letelier, I. Ramos, P. Sánchez, and O. Pastor. *OASIS 3.0: Un Enfoque Formal para el Modelado Conceptual Orientado a Objetos*. SPUPV-98.4011, Universidad Politécnica de Valencia, España, 1998.

[LSRI00] J. Lee, V. Su, S. Ren, and H. Ishii. HandSCAPE: a vectorizing tape measure for on-site measuring applications. In *Proc. of CHI 2000*, pages 137–144, 2000.

[LSZB98] J.P. Loyall, R.E. Schantz, J.A. Zinky, and D.E. Bakken. Specifying and measuring quality of service in distributed object systems. In *Proc. of the 1st IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'98)*, page 43, Washington, DC, USA, 1998. IEEE Computer Society. ISBN: 0-8186-8430-5.

[Luy04] K. Luyten. *Dynamic User Interface Generation for Mobile and Embedded Systems with Model-Based User Interface Development*. PhD thesis, Limburgs Universitair Centrum, transnational University Limburg, School of Information Technology, Expertise Centre for Digital Media, Diepenbeek, Belgium, 2004.

[LV04] Q. Limbourg and J. Vanderdonckt. UsiXML: A User Interface Description Language Supporting Multiple Levels of Independence. In M. Matera and S. Comai, editors, *Engineering Advanced Web Applications*. Rinton Press, Paramus, 2004. Collection of papers presented at the Workshops IWWOST, AHCW, WQ, Device Independent Web Engineering (DIWE'04), International Conference on Web Engineering (ICWE'04).

[LVM+04] Q. Limbourg, J. Vanderdonckt, B. Michotte, L. Bouillon, and V.M. López-Jaquero. UsiXML: a Language Supporting Multi-Path Development of User Interfaces. In R. Bastide, P.A. Palanque, and J. Roth, editors, *Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems (EHCI-DSVIS'04)*, volume 3425 of *Lecture Notes in Computer Science*, pages 200–220. Springer-Verlag - Berlin, 2004.

[LVS00] Q. Limbourg, J. Vanderdonckt, and N. Souchon. The Task-Dialog and Task-Presentation Mapping Problem: Some Preliminary Results. In P.A. Palanque and F. Paternò, editors, *Proc. of the 7th International Workshop on Design, Specification, Verification of Interactive Systems (DSV-IS'00)*, volume 1946, pages 227–246, Berlin, ALLEMAGNE, 2000. Springer. ISBN 3-540-41663-3.

[Mae87] P. Maes. Concepts and experiments in computational reflection. In *Proc. of the 2nd Conference Object-oriented programming systems, languages and applications (OOPSLA'87)*, pages 147–155, New York, NY, USA, 1987. ACM Press. ISBN: 0-89791-247-0.

[Mär90] Ch. Märtin. A UIMS for Knowledge Based Interface Template Generation and Interaction. In *Proc. of the IFIP TC13 Third Interational Conference on Human-Computer Interaction (INTERACT'90)*, pages 651–657, Amsterdam, The Netherlands, 1990. North-Holland Publishing Co. ISBN: 0-444-88817-9.

[MB02] V. Marangozova and F. Boyer. Using reflective features to support mobile users. In W. Cazzola, S. Chiba, and T. Ledoux, editors, *On-Line Proc. of ECOOP'2000 Workshop on Reflection and Metalevel Architectures*, 2002. http://www.disi.unige.it/person/CazzolaW/ewrma2000-proc..html.

[MBdL02] C. Mesquita, S.D.J. Barbosa, and C.J.P. de Lucena. Towards the identification of concerns in personalization mechanisms via scenarios. *Workshop on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, held in conjunction with AOSD'02 Conference*, 2002.

[MdL01] M.E. Markiewicz and C.J.P. de Lucena. El desarrollo del framework orientado al objeto. *ACM Crossroads Student Magazine, The ACM's First Electronic Publication*, La Ingenieria del Sofware(7.4), 2001. Traducido por Paulo N. Lama.

[MHP00] B. Myers, S. Hudson, and R. Pausch. Past, Present, and Future of User Interface Software Tools. *ACM Transactions Computer-Human Interaction*, 7(1):3–28, 2000. ACM Press.

[MLGR02] F. Montero, M.D. Lozano, P. González, and I. Ramos. A first approach to design Web sites by using patterns. In P. Hruby and K.E. Sorensen, editors, *Proc. of the First Nordic Conference on Pattern languages of Programs (Viking PLoP'02)*, pages 137–158, Copenhagen, Denmark, 2002. Microsoft Business Solutions, ApS. ISBN: 87-7849-769-8.

[MLLG06] F. Montero, V. López-Jaquero, M. Lozano, and P. González. *Selection of HCI related papers of Interacción 2004*, chapter A user interfaces development and abstraction mechanisms, pages 329–336. Computer Science. Springer-Verlag, 2006. ISBN: 978-1-4020-4204-1.

[MLML03] F. Montero, V. López-Jaquero, J.P. Molina, and M.D. Lozano. Improving e-Shops Environments by Using Usability Patterns. In *Proc. of the 2nd Workshop on Software and Usability Cross-Pollination: the role of usability patterns. Official Workshop of IFIP working group 13.2. (INTERACT 2003)*, Zürich, Switzerland, 2003.

[MLV+05] F. Montero, V. López-Jaquero, J. Vanderdonckt, P. González, M.D. Lozano, and Q. Limbourg. Solving the mapping problem in user interface design by seamless integration in idealxml. In S. Gilroy and M. Harrison, editors, *Proc. of 12th International Workshop on Design, Specification and Verification of Interactive Systems (DSV-IS'05)*, number 3941 in Lecture Notes in Computer Science, pages 161–172, Berlin, Germany, 2005. Springer-Verlag.

[MMG05] N. Medina, F. Molina, and L. García. Diversity of structures and adaptive methods on an evolutionary hypermedia system. *IEE Proc.-Software*, 152(3):119–126, 2005.

[MMM+97] B.A. Myers, R.G. McDaniel, R.C. Miller, A.S. Ferrency, A. Faulring, B.D. Kyle, A. Mickish, A. Klimovitski, and P. Doane. The amulet environment: New models for effective user interface software development. *IEEE Trans. Softw. Eng.*, 23(6):347–365, 1997. IEEE Press.

[Mol04] P.J. Molina. A Review to Model-Based User Interface Development Technology. In H. Trætteberg, P.J. Molina, and N.J. Nunes, editors, *Proc. of the First International Workshop on Making model-based user interface design practical: usable and open methods and tools*, volume 103 of *CEUR Workshop Proceedings*, Funchal, Madeira, Portugal, 2004. http://CEUR-WS.org/Vol-103.

[Mol07] A.I. Molina. *Una propuesta metodológica para el desarrollo de la interfaz de usuario en sistemas groupware.* PhD thesis, Escuela Superior de Informática, Universidad de Castilla-La Mancha, 2007.

[MP02] L. Marucci and F. Paternò. Supporting Adaptivity for Heterogeneous Platforms through User Model. In *Proc. of the 4th International Symposium on Mobile Human-Computer Interaction (Mobile HCI'02)*, pages 409–413, London, UK, 2002. Springer-Verlag. ISBN: 3-540-44189-1.

[MPI05] MPIu+a. Modelo de Proceso de la Ingeniería de la Usabilidad y de la Accesibilidad. http://griho.udl.es/mpiua, 2005. ISBN: 8497883209.

[MPS02] G. Mori, F. Paternò, and C. Santoro. CTTE: support for developing and analyzing task models for interactive system design. *IEEE Transaction Software Engineering*, 28(8):797–813, 2002.

[MPS03] G. Mori, F. Paternò, and C. Santoro. Tool support for designing nomadic applications. In *Proc. of the International Conference on Intelligent User Interfaces (IUI'03)*, pages 141–148, New York, NY, USA, 2003. ACM Press. ISBN: 1-58113-586-6.

[MPS04] G. Mori, F. Paternò, and C Santoro. Design and development of Multidevice User Interfaces through Multiple Logical Descriptions. *IEEE Transactions on software engineering*, 30(8):507–520, 2004. IEEE Press.

[MS97] R. Majan and B. Shneiderman. Visual and Textual Consistency Checking Tools for Graphical User Interfaces. *IEEE Trans. Softw. Eng.*, 23(11):722–735, 1997. IEEE Press.

[MVG06] J. P. Molina, J. Vanderdonckt, and P. González. Direct manipulation of user interfaces for migration. In *Proc. of the 11th international conference on Intelligent user interfaces (IUI'06)*, pages 140–147, New York, NY, USA, 2006. ACM Press. ISBN: 1-59593-287-9.

[MWB+01] G.C. Murphy, R.J. Walker, E.L.A. Baniassad, M.P. Robillard, A. Lai, and M.A. Kersten. Does aspect-oriented programming work? *Communications of the ACM*, 44(10):75–77, 2001.

[MY93] S. Matsuoka and A. Yonezawa. Analysis of inheritance anomaly in object-oriented concurrent programming languages. *Research directions in concurrent object-oriented programming*, pages 107–150, 1993. ISBN: 0-262-01139-5.

[NAMA01] J. Nicola, M. Abney, M. Mayfield, and M. Abney. *Streamlined Object Modeling: Patterns, Rules, and Implementation with Cdrom.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[NC85] A. Newell and S.K. Card. The prospects for psychological science in human-computer interaction. *Human-Computer Interaction*, 1(3):209–242, 1985.

[NC00] N. Nunes and J. Cunha. Wisdom: A UML Based Architecture for Interactive Systems. In Ph. Palanque and F. Paternò, editors, *Proc. of 7th International Workshop (DSV-IS)*, volume 1946, pages 191–205, Limerick, Ireland, 2000. Springer.

[Nel98] G.J. Nelson. *Context-aware and location systems.* Phd dissertation, University of Cambridge, United Kingdom, 1998.

[Nie93] J. Nielsen. Usability Engineering. Academic Press, Boston, 1993. ISBN: 0-12-518405-0.

[NOP06] A. Neyem, S.F. Ochoa, and J.A. Pino. Supporting Mobile Collaboration with Service-Oriented Mobile Units. In *Proc. of CRIWG, LNCS 4154*, pages 228–245. Springer, 2006.

[Nor86] Draper S.W. Norman, D.A. *User Centered System Design; New Perspectives on Human-Computer Interaction.* Lawrence Erlbaum Associates, Inc., Mahwah, NJ, USA, 1986.

[OJN+00] D. Olsen, S. Jefferies, T. Nielsen, W. Moyes, and O. Fredichson. Cross-modal Interaction Using XWeb. In *Proc. of the 13th annual ACM symposium on User interface software and technology (UIST'00)*, pages 191–200, New York, NY, USA, 2000. ACM Press. ISBN: 1-58113-212-3.

[OMT98] P. Oreizy, N. Medvidovic, and R.N. Taylor. Architecture-Based Runtime Software Evolution. In *Proc. of the 20th international conference on Software engineering (ICSE'98)*, pages 177–186, Washington, DC, USA, 1998. IEEE Computer Society. ISBN: 0-8186-8368-6.

[OR99] L. Ora and S. Ralph. Resource Description Framework (RDF). http://www.w3c.org/TR/1999/REC-rdf-syntax, 1999. Model and Syntax Specification. W3C Recommendation.

[OS00] R. Oppermann and M. Specht. A Context-Sensitive Nomadic Exhibition Guide. In *Proc. of the 2nd international symposium on Handheld and Ubiquitous Computing (HUC'00)*, pages 127–142, London, UK, 2000. Springer-Verlag. ISBN 3-540-41093-7.

[PAF00] O. Pastor, S. Abrahäo, and J.J. Fons. An Object-Oriented approach for Web-solutions Modelling. In *Proc. of MEdia in Information Society (MEIS'00)*, pages 127–128, 2000.

[Pas97] J. Pascoe. The stick-e note architecture: extending the interface beyond the user. In *Proc. of the 2nd international conference on Intelligent user interfaces (IUI'97)*, pages 261–264, New York, NY, USA, 1997. ACM Press. ISBN: 0-89791-839-8.

[Pas98] J. Pascoe. Adding generic contextual capabilities to wearable computers. In *Proc. of the 2nd IEEE International Symposium on Wearable Computers (ISWC'98)*, pages 92–99, Washington, DC, USA, 1998. IEEE Computer Society. ISBN: 0-8186-9074-7.

[Pat99] F. Paternò. *Model-Based Design and Evaluation of Interactive Applications.* Springer-Verlag, London, UK, 1st edition, 1999.

[PCD92] R. Pausch, M. Conway, and R. DeLine. Lessons learned from SUIT, the Simple User Interface Toolkit. *ACM Transactions on Office Information Systems*, 10(4):320–344, 1992.

[PE99] A. Puerta and J. Eisenstein. Towards a General Computational Framework for Model-Based Interface Development Systems. In *Proc. of the 4th international*

Conference on Intelligent user interfaces (IUI'99), pages 171–178, New York, NY, USA, 1999. ACM Press.

[PE02] A. Puerta and J. Eisenstein. XIML: A Common Representation for Interaction Data. In *Proc. of the 7th international conference on Intelligent user interfaces (IUI'02)*, pages 214–215, New York, NY, USA, 2002. ACM Press. ISBN: 1-58113-459-2.

[PEGM94] A. R. Puerta, H. Eriksson, J. H. Gennari, and M. A. Musen. Model-based automated generation of user interfaces. In AAAI Press, editor, *Proc. of the 12th National Conference on Artificial Intelligence (AAAI'94)*, volume 1, pages 471–477, Menlo Park, CA, USA, 1994. American Association for Artificial Intelligence. ISBN: 0-262-61102-3.

[PFT02] M. Pinto, L. Fuentes, and J.M. Troya. Plataforma para la composición dinámica de componentes y aspectos. In M. Celma, O. Pastor, N.J. Juzgado, and J.J. Moreno-Navarro, editors, *Proc. of VII Jornadas de Ingeniería del Software y Bases de Datos (JISBD'02)*, pages 387–398, 2002. ISBN: 84-688-0206-9.

[PHK03] M. Panahi, T. Harmon, and R. Klefstad. Adaptive techniques for minimizing middleware memory footprint for distributed, real-time, embedded systems. In *Proc. of the IEEE 18th Annual Workshop on Computer Communications (CCW'03)*, volume 18, pages 54–58. eScholarship-Repository, Postprints, paper 656: University of California, 2003. ISBN: 0-7803-8239-0.

[Pin00] P. Pinheiro da Silva. User Interface Declarative Models and Development Environments: A Survey. *Lecture Notes in Computer Science*, 1946:207–226, 2000.

[Pin02] P. Pinheiro da Silva. *Object modelling of interactive systems: The UMLi approach.* PhD thesis, Department of Computer Science, University of Manchester, United Kingdom, 2002.

[PIP+97] O. Pastor, E. Isfrán, V. Pelechano, J. Romero, and J. Meseguer. OO-Method: An OO Software production environment combining conventional and formal methods. In A. Olivé and J.A. Pastor, editors, *Proc. of CAiSE'97*, volume 1250 of *LNCS*, pages 145–158. Springer-Verlag, 1997. ISBN: 3-540-63107-0.

[Pir02] V Piroumian. *Wireless J2me Platform Programming.* Sun Microsystems, Java series. Prentice Hall Professional Technical Reference, 1st. edition, 2002. Foreword By-Mike Clary and Foreword By-Bill Joy.

[PL97] A.I. Periquet and E. Lin. Mobility reflection: Exploiting mobility-awareness in applications by reflecting on distributed object collaborations. Technical report 97-CSE-6, Department of Computer Science and Engineering of Southern Methodist University, Dallas, TX., 1997. http://citeseer.ist.psu.edu/update/94248.

[PLF+01] S. Ponnekanti, B. Lee, A. Fox, P. Hanrahan, and T. Winograd. Icrafter: a service framework for ubiquitous computing environments. In G. Abowd, B. Brumitt, and S. Shafer, editors, *Proc. of Ubicomp*, volume 2201 of *LNCS*, pages 57–75, 2001. Springer-Verlag.

[PLL04] C. Paris, S. Lu, and K. V. Linden. Environments for the construction and use of task models. In D. Diaper and N. A. Stanton, editors, *The Handbook of Task Analysis for Human-Computer Interaction*, chapter 23, pages 467–482. Lawrence Erlbaum Associates, Mahwah, NJ, 2004. http://www.calvin.edu/~kvlinden/distributions/Paris-VL-Lu-HTA-2003.pdf.

[PLV01] C. Pribeanu, Q. Limbourg, and J. Vanderdonckt. Task Modelling for Context-Sensitive User Interfaces. In *Proc. of the 8th International Workshop on Interactive Systems: Design, Specification, and Verification (DSV-IS'01)*. Springer-Verlag London, UK, 2001. ISBN: 3-540-42807-0.

[PMM97] F. Paternò, C. Mancini, and S. Meniconi. ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In *Proc. of the IFIP TC13 Interantional Conference on Human-Computer Interaction (INTERACT'97)*, pages 362–369. Chapman & Hall, 1997. ISBN: 0-412-80950-8.

[PP00] P. Pinheiro da Silva and N.W. Paton. UMLi: The Unified Modeling Language for Interactive Applications. In A. Evans, S. Kent, and B. Selic, editors, *Proc. of The Unified Modeling Language, Advancing the Standard, Third International Conference (UML)*, volume 1939 of *Lecture Notes in Computer Science*, pages 117–132, York, UK, 2000. Springer-Verlag.

[PP02] L. Paganelli and F. Paternò. Automatic reconstruction of the underlying interaction design of web applications. In *Proc. of the 14th international conference on Software engineering and knowledge engineering (SEKE'02)*, pages 439–445, New York, NY, USA, 2002. ACM Press. ISBN: 1-58113-556-4.

[PR96] K.A. Palfreyman and T. Rodden. A protocol for user awareness on the World Wide Web. In *Proc. of the ACM Conference on Computer Supported Cooperative Work (CSCW'96)*, pages 130–139, New York, NY, USA, 1996. ACM Press. ISBN: 0-89791-765-0.

[PRM99] J. Pascoe, N.S. Ryan, and D.R. Morse. Issues in developing context-aware computing. In *Proc. of the International Symposium on Handheld and Ubiquitous Computing*, pages 208–221. Springer-Verlag, 1999.

[PRS+94] J. Preece, Y. Rogers, H. Sharp, D. Benyon, S. Holland, and T. Carey. *Human-Computer Interaction.* Addison-Wesley Longman Ltd., Essex, UK, 1994. ISBN 0-201-62769-8.

[PS03] F. Paternò and C. Santoro. A Unified Method for Designing Interactive Systems Adaptable to Mobile and Stationary Platforms. *Interacting with Computers. Elsevier*, 15(3):347–364, 2003.

[Pue96] A.R. Puerta. The MECANO Project: Comprehensive and Integrated Support for Model-Based Interface Development. In J. Vanderdonckt, editor, *Proc. of Computer-Aided Design of User Interfaces (CADUI'96)*, pages 19–25. Presses Universitaires de Namur. Namur, Belgium, 1996.

[Pue97] A. R. Puerta. A Model-Based Interface Development Environment. *IEEE Software*, 14(4):40–47, 1997. IEEE Computer Society Press.

[Pue98] A.R. Puerta. Supporting user-centred design of adaptive user interfaces via interface models. In *Proc. of 1rst annual Workshop on Real-Time Intelligent User Interfaces for Decision Support and Information Visualization*, 1998. http://www-smi.stanford.edu/pubs/SMI_Reports/SMI-98-0747.pdf.

[RBIM98] D. Roberts, D. Berry, S. Isensee, and J. Mullaly. *Designing for the user with OVID: Bridging user interface design and software engineering.* Software Engineering Series. MacMillan Technical Publishing, Indianapolis, 1998. ISBN 10: 1578701015 - 13: 978-1578701018.

[RC04] G. Rey and J. Coutaz. Contextor: capture and dynamic distribution of contextual information. In *Proc. of the 1st French-speaking conference on Mobility and ubiquity computing: UbiMob'04*, pages 131–138, New York, NY, USA, 2004. ACM Press. ISBN: 1-58113-915-2.

[Rei00] A. M. Reina-Quintero. Visión general de la programación orientada a aspectos. informes, estudios, trabajos y dictámenes. Artítulo técnico LSI-2000-11, Facultad de Informática y Estadística. Departamento de Lenguajes y Sistemas Informáticos. Universidad de Sevilla, Spain, 2000.

[Rek00] J. Rekimoto. Multiple-Computer User Interfaces: Beyond the desktop. In *Proc. of Human Factors in Computing Systems (CHI'00)*, pages 6–7, New York, NY, USA, 2000. ACM Press. ISBN: 1-58113-248-4.

[RN01] P. Renevier and L. Nigay. Mobile Collaborative Augmented Reality: The Augmented Stroll. In *Proc. of the 8th IFIP International Conference on Engineering for Human-Computer Interaction (EHCI'01)*, volume 2254, pages 299–316, London, UK, 2001. Springer-Verlag.

[Rot02] J. Roth. Patterns of Mobile Interaction. *Personal and Ubiquitous Computing*, 6(4):282–289, 2002. Springer-Verlag, London, UK.

[Roz97] G. Rozenberg. *Handbook of graph grammars and computing by graph transformation*, volume 1 of *Foundations*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1997.

[RPM97] N.S. Ryan, J. Pascoe, and D.R. Morse. Enhanced reality fieldwork: the context-aware archaeological assistant. In L. Dingwall, S. Exon, V. Gaffney, S. Laflin, and M. Van Leusen, editors, *Proceeding of the 25th Anniversary (CAA'97), Computer Applications in Archaeology: Archaeology in the Age of the Internet, held at the University of Birmingham*, number Section 11: Finds Analysis and Curation in British Archaeological Reports, Oxford, UK, 1997. The BAR International Series 750, Archaeopress. http://www.cs.kent.ac.uk/projects/mobicomp/Fieldwork/Papers.

[Sap04] A.M. Sapienza. *Managing Scientists: Leadership Strategies in Scientific Research*. Wiley-IEEE, 2nd. edition, 2004.

[SAT+99] A. Schmidt, K.A. Aidoo, A. Takaluoma, U. Tuomela, K. Van Laerhoven, and W. Van de Velde. Advanced interaction in context. In *Proc. of the 1st international symposium on Handheld and Ubiquitous Computing (HUC'99)*, pages 89–101, London, UK, 1999. Springer-Verlag. ISBN: 3-540-66550-1.

[SAW94] B. Schilit, N. Adams, and R. Want. Context-Aware Computing Applications. In *Proc. of IEEE Workshop on Mobile Computing Systems and Applications*, pages 85–90, 1994.

[SB02] S. Soares and P. Borba. PaDA: A pattern for distribution aspects. In *Proc. of Second Latin American Conference on Pattern Languages of Programming (SugarLoafPLoP'02)*, 2002.

[SC02] K.A. Schneider and J.R. Cordy. AUI: A Programming Language for Developing Plastic Interactive Software. In *Proc. of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)*, volume 9, pages 3656–3665, 2002.

[SC06] M. Sendín and C.A. Collazos. Implicit plasticity framework: A client-side generic framework for collaborative activities. In Y.A. Dimitriadis, I. Zigurs, and E. Gómez-Sánchez, editors, *Proc. of Groupware: Design, Implementation, and Use, 12th International Workshop (CRIWG'06)*, volume 4154 of *Lecture Notes in Computer Science*, pages 219–227. Springer, 2006. ISBN: 3-540-39591-1.

[Sch94] S. Schreiber. The boss-system: Coupling visual programming with model based interface design. In F. Paterno, editor, *Proc. of the First International Eurographics Workshop. Design, Specification and Verification of Interactive Systems'94 (DSV-IS)*, pages 161–179. Springer-Verlag, 1994. ISBN: 3-540-59480-9.

[Sch95] W.N. Schilit. *A system architecture for context-aware mobile computing.* Thesis dissertation, Columbia University, New York, NY, USA, 1995. UMI Order No. GAX95-33659.

[Sch96] E. Schlungbaum. Model-Based User Interface Software Tools. Current state of Declarative Models. current state of declarative models. Technical Report 96–30, GIT-GVU, 1996.

[Sch06] D.C. Schmidt. Introduction to patterns and frameworks. http://www.cs.wustl.edu/, 2006. Tutorial of the Department of EECS, Vanderbilt University.

[SDA98] D. Salber, A.K. Dey, and G.D. Abowd. Ubiquitous computing: Defining an hci research agenda for an emerging interaction paradigm. Tech. report git-gvu-98-01, IFIP Working Conference on Engineering for Human-Computer Interaction. Georgia Institute of Technology, Atlanta, GA, USA, 1998.

[SDA99] D. Salber, A.K. Dey, and G.D. Abowd. The context toolkit: aiding the development of context-enabled applications. In *Proc. of the SIGCHI conference on Human factors in computing systems (CHI'99)*, pages 434–441, New York, NY, USA, 1999. ACM Press. url: http://www.cc.gatech.edu/fce/contexttoolkit/pubs/SEWPC00.pdf.

[SE96a] E. Schlungbaum and T. Elwert. Automatic user interface generation from declarative models. In J. Vanderdonckt, editor, *Computer-Aided Design of User Interfaces I, Proceedings of the 2nd International Workshop on Computer-Aided De-*

sign of User Interfaces (CADUI'96), pages 3–18, Namur, Belgium, 1996. Presses Universitaires de Namur. ISBN: 2-87037-232-9.

[SE96b] E. Schlungbaum and T. Elwert. Dialogue Graphs: A Formal and Visual Specification Technique for Dialogue Modelling. In C.R. Roast and J.I. Siddiqi, editors, *Proc. of the BCS-FACS Workshop on Formal Aspects of the Human Computer Interface - Sheffield Hallam University*. Springer-Verlag London, 1996.

[Sen07] M. Sendín. *Revisión del Estado del Arte en Computación Ubicua*. Escuela Politécnica Superior, Universitat de Lleida, 2007. Apuntes de la asignatura de Tecnologías de Interacción. Estilos y Paradigmas. Master en IPO.

[SER+04] V. Sacramento, M. Endler, H.K. Rubinsztejn, L.S. Lima, K.M. Gonçalves, F.N. Nascimento, and G.A. Bueno. Moca: A middleware for developing collaborative applications for mobile users. *IEEE Distributed Systems Online*, 5(10), 2004.

[SG02] J.P. Sousa and D. Garlan. Aura: an Architectural Framework for User Mobility in Ubiquitous Computing Environments. In J. Bosch, M. Gentleman, C. Hofmeister, and J. Kuusela, editors, *Proc. of the 3rd Working IEEE/IFIP Conference on Software Architecture*, volume 224 of *IFIP Conference Proc.*, pages 29–43, Montréal, Québec, Canada, 2002. Kluwer Academic Publishers. ISBN: 1-4020-7176-0.

[Shn92] B. Shneiderman. *Designing the user interface. Strategies for effective Human-Computer Interaction*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 1992.

[SKSH96] C. Schuckmann, L. Kirchner, J. Schümmer, and J.M. Haake. Designing object-oriented synchronous groupware with coast. In *Proc. of the 1996 ACM conference on Computer supported cooperative work (CSCW'96)*, pages 30–38, New York, NY, USA, 1996. ACM Press. ISBN: 0-89791-765-0.

[SL04] M. Sendín and J. Lorés. Plasticidad implícita en dispositivos móviles: Hacia la ortogonalidad deseada en la separación de conceptos. In *Proc. of V Congreso en Interacción Persona-Ordenador (Interacción 2004)*, pages 70–78, 2004. ISBN: 84-609-1266-3.

[SL05a] M. Sendín and J. Lorés. *Selection of HCI related papers of Interacción 2004*, chapter Remote Support to Plastic User Interfaces: a Semantic View, pages 55–70. Computer Science. Springer-Verlag Netherlands, 1st edition, 2005. ISBN 13: 978-1-4020-4204-1.

[SL05b] M. Sendín and J. Lorés. *Selection of HCI related papers of Interacción 2004*, chapter Local Support to Plastic User Interfaces: an Orthogonal Approach, pages 163–168. Computer Science. Springer-Verlag Netherlands, 1st edition, 2005. ISBN: 978-1-4020-4204-1.

[SLF03] T. Strang, C. Linnhoff-Popien, and K. Frank. Cool: A context ontology language to enable contextual interoperability. In I.M. Dameure J.-B. Stefani and D. Hagimont, editors, *Proc. of 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS'03)*, volume 2893 of *Lecture Notes in Computer Science*, pages 236–247. Springer Verlag, 2003. ISBN: 978-3-540-20529-6.

[SLP04] T. Strang and C. Linnhoff-Popien. A context modeling survey. In *Proc. of First International Workshop on Advanced Context Modelling, Reasoning and Management as part of (UbiComp'04)-The Sixth International Conference on Ubiquitous Computing*, 2004.

[SLV02] N. Souchon, Q. Limbourg, and J. Vanderdonckt. Task Modelling in Multiple Contexts of Use. In *Proc. of the 9th International Workshop on Interactive Systems. Design, Specification, and Verification (DSV-IS'02)*, volume 2545 of *LNCS*, pages 59–73, London, UK, 2002. Springer-Verlag. ISBN: 3-540-00266-9.

[SM86] S.L. Smith and J.N. Mosier. Guidelines for designing user interface software. Computer Programming and Software MTR-10090, The MITRE Corporation, Bedford, Massachusetts, USA, 1986.

[SMC+07] M. Sendín, N. Medina, M. Cabrera, V. López-Jaquero, and L. García. Integración de mecanismos de adaptación de contenidos y generación de ius plásticas. In *Actas del V Taller en Sistemas Hipermedia Colaborativos y Adaptativos (SHCA'07), en conjunción con el 2º Congreso Español De Informática (CEDI 2007)*, 2007.

[Smi84] B.C. Smith. Reflection and semantics in lisp. In *Proc. of the 11th ACM SIGACT-SIGPLAN symposium on Principles of programming languages: (POPL'84)*, pages 23–35, New York, NY, USA, 1984. ACM Press. ISBN: 0-89791-125-3.

[SMR+97] T. Starner, S. Mann, B.J. Rhodes, J. Levine, J. Healey, D. Kirsch, R.W. Picard, and A. Pentland. Augmented reality through wearable computing. *Presence*, 6(4):386–398, 1997.

[Som05] I. Sommerville. *Software engineering.* Addison Wesley, 7th edition, 2005.

[SP90] D. Scapin and C. Pierret-Golbreich. Towards a method for task description: MAD. In *Proc. of Work with Display Unit'89*, pages 371–80. Elsevier Science, 1990.

[SS00] N. Sawhney and C. Schmandt. Nomadic radio: speech and audio interaction for contextual messaging in nomadic environments. *ACM Transaction Computer-Human Interaction (ToCHI)*, 7(3):353–383, 2000.

[SSC+96] P.A. Szekely, P.N. Sukaviriya, P. Castells, J. Muthukumarasamy, and E. Slacher. Declarative interface models for user interface construction tools: the MASTERMIND approach. In *Proc. of the IFIP TC2/WG2.7 Working Conference on Engineering for Human-Computer Interaction*, pages 120–150, London, UK, UK, 1996. Chapman & Hall, Ltd. ISBN: 0-412-72180-5.

[Sut97] A. Sutcliffe. Task-related information analysis. *International Journal of Human-Computer Studies*, 47(2):223–257, 1997.

[SV03] N. Souchon and J. Vanderdonckt. A review of xml-compliant user interface description languages. In J.A. Jorge, N.J. Nunes, and J.F.Cunha, editors, *Proc. of the 10th International Workshop, Interactive Systems. Design, Specification, and Verification (DSV-IS'03)*, volume 2844 of *Lecture Notes in Computer Science*, pages 377–391. Springer, 2003. ISBN: 3-540-20159-9.

[SV06] M. Sendín and J. Viladrich. Contrasting aspectual decompositions with object-oriented designs in contexts-aware mobile applications. In *Workshop en Desarrollo de Software Orientado a Aspectos (DSOA'06), in conjunction with the XV Jornadas de Ingeniería del Software y Bases de Datos*, 2006.

[Sze96] P. Szekely. Retrospective and challenges for model-based interface development. In F. Bodart and J. Vanderdonckt, editors, *Proc. of the 2nd International Workshop on Computer-Aided Design of User Interfaces. Design, Specification and Verification of Interactive Systems (DSV'96)*, pages 1–27, Namur University Press, 1996. Springer-Verlag.

[TC99] D. Thevenin and J. Coutaz. Plasticity of User Interfaces: Framework and Research Agenda. In *Proc. of INTERACT'99, IFIP TC.13 Conference on Human–Computer Interaction*, volume 1, pages 110–117, Edinburgh, UK, 1999. IOS Press.

[The01] D. Thevenin. *Adaptation en Interaction Homme-Machine: Le cas de la Plasticité.* PhD thesis, Joseph Fourier University, Grenoble, France, 2001.

[Tid02] J. Tidwell. *Designing Interfaces: Patterns for Effective Interaction Design.* O'Reilly Media, Inc., 2002.

[TOHS99] P. Tarr, H. Ossher, W. Harrison, and S.M. Sutton. N degrees of separation: multi-dimensional separation of concerns. In *Proc. of the 21st international conference on Software engineering: (ICSE'99)*, pages 107–119, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press. ISBN: 1-58113-074-0.

[Van95] J. Vanderdonckt. Knowledge-Based Systems for Automated User Interface Generation: the TRIDENT Experience. Technical Report RP-95-010, Facultés Universitaires Notre-Dame de la Paix, Institut d'Informatique, Namur, Belgium, 1995.

[Van97] Jean Vanderdonckt. *Conception assistée de la présentation d'une interface homme-machine ergonomique pour une application de gestion hautement interactive*. PhD thesis, Facultes Universitaires Notre-Dame de la Paix, Institut d'Informatique, Namur, Belgium, 1997. FUNDP, Institut d'Informatique.

[Van99] Jean Vanderdonckt. Assisting designers in developing interactive business oriented applications. In Hans-Jörg Bullinger and Jürgen Ziegler, editors, *Proc. of 8th Int. Conf. on Human-Computer Interaction of HCI International'99*, volume 1, pages 1043–1047. Lawrence Erlbaum, 1999. ISBN: 0-8058-3391-9.

[VB93] J. Vanderdonckt and F. Bodart. Encapsulating Knwoledge for Intelligent Automatic Interaction Objects Selection. In *Proc. of the SIGCHI conference on Human factors in computing systems (CHI'93)*, pages 424–429, New York, NY, USA, 1993. ACM Press. ISBN: 0-89791-575-5.

[VB99] J. Vanderdonckt and P. Berquin. Towards a Very Large Model-Based Approach for User Interface Development. In *Proc. of the 1999 User Interfaces to Data Intensive Systems (UIDIS'99)*, pages 76–85, Washington, DC, USA, 1999. IEEE Computer Society. ISBN: 0-7695-0262-8.

[VCSH04] J. Vassileva, R. Chen, L. Sun, and W. Han. Stimulating user participation in a File-Sharing P2P system supporting university classes. *P2P*, 2004.

[VFOM01] J. Vanderdonckt, M. Florins, F. Ogen, and B. Macq. Model-Based Design of Mobile User Interfaces. In M.D. Dunlop and S.A. Brewster, editors, *Proc. of 3rd International Workshop on Human Computer Interaction with Mobile Devices (Mobile HCI'01)*, pages 135–140, 2001.

[Vil06] J. Viladrich. Disseny i implementació de components software per dotar de personalització i sensibilitat al context a aplicaciones mòbils utilitzant programació orientada a aspectes. Trabajo de final de carrera de ingeniería en informática, Escuela Politécnica Superior, Universitat de Lleida, 2006.

[VLF03] J. Vanderdonckt, Q. Limbourg, and M. Florins. Deriving the Navigational Structure of a User Interface. In M. Rauterberg, M. Menozzi, and J. Wesson, editors, *Proc. of the 9th IFIP TC 13 Int. Conference on Human-Computer Interaction (INTERACT'03)*, pages 455–462, Zürich, 2003. IOS Press.

[Was85] A. Wasserman. Extending state transition diagrams for the specification of human-computer interaction. *IEEE Transactions on software engineering*, 11(8):699–713, 1985. IEEE Press.

[Wei91] M. Weiser. *The Computer for the Twenty-First Century*, volume 265, pages 94–104. Scientific American, 1991. http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html.

[Wei93] M. Weiser. *Some Computer Science Problems in Ubiquitous Computing*, pages 137–143. Communications of the ACM. Nikkei Electronics, 1993. reprinted as Ubiquitous Computing.

[Wel04] M. van Welie. Patterns in interaction design. http://www.welie.com/patterns/literature.html, 2004.

[WHFG92] R. Want, A. Hopper, V. Falcäo, and J. Gibbons. The active badge location system. *ACM Trans. Inf. Syst.*, 10(1):91–102, 1992. ACM Press.

[Wik07] Wikipedia Foundation, Inc. Wikipedia. The free encyclopedia. http://en.wikipedia.org/wiki/Wikipedia, 2007.

[WK99] J. Warmer and A. Kleppe. *The Object Constraint language: Precise Modelling with UML*. Object Technology Series. Addison-Wesley Professional, Massachusetts, 1st edition, 1999.

[WLF00] M. Wermelinger, A. Lopes, and J.L. Fiadeiro. Superposing connectors. In *Proc. of the 10th International Workshop on Software Specification and Design (IWSSD'00)*, pages 87–94, Washington, DC, USA, 2000. IEEE Computer Society. ISBN: 0-7695-0884-7.

[WSA+95] R. Want, B.N. Schilit, N.I. Adams, R. Gold, K. Petersen, D. Goldberg, J.R. Ellis, and M. Weiser. An overview of the PARCTAB ubiquitous computing experiment. *IEEE Personal Communications*, 2(6):28–33, 1995.

[Xer00] Xerox Corporation. The AspectJTM Programming Guide. http://www.eclipse.org/aspectj/doc/released/progguide/index.html, 2000. Palo Alto Research Center.

[YJ02] J.W. Yoder and R. E. Johnson. The adaptive object-model architectural style. In J. Bosch, W. M. Gentleman, C. Hofmeister, and J. Kuusela, editors, *Proc. of Software Architecture: System Design, Development and Maintenance, IFIP World Computer Congress - Conference on Software Architecture (WICSA3)*, volume 224 of *IFIP Conference Proc.*, pages 3–27. Kluwer, 2002. ISBN: 1-4020-7176-0.

[YL04] L. Yamane and J. Lorés. Els Vilars: A Cultural Heritage Augmented Reality Device. In *Proc. of V Congreso en Interacción Persona-Ordinador (Interacción'04)*, pages 62–69, 2004. ISBN: 84-609-1266-3.

[You05] T.J. Young. Using AspectJ to build a software product line for mobile devices. Master's thesis, University of British Columbia, 2005. Master of Science in Computer Science.

[YS00] H. Yan and T. Selker. Context-aware office assistant. In *Proc. of the 5th international conference on Intelligent user interfaces (IUI'00)*, pages 276–279, New York, NY, USA, 2000. ACM Press. ISBN: 1-58113-134-8.

[Yu97] E. Yu. Towards modelling and reasoning support for early-phase requirements engineering. In *Proc. of the 3rd IEEE International Symposium. On Requirements Engineering (RE'97)*, pages 226–235. IEEE Computer Society , Washington, DC, USA, 1997. ISBN: 0-8186-7740-6.

[Yu04] E. Yu. University of Toronto Canada, 2004. GRL (Goal-Oriented Requirement Language).

[ZGJ04] A. Zambrano, S.E. Gordillo, and I. Jaureguiberry. Aspect-based adaptation for ubiquitous software. In F. Crestani, M.D. Dunlop, and S. Mizzaro, editors, *Mobile HCI Workshop on Mobile and Ubiquitous Information Access*, volume 2954 of *Lecture Notes in Computer Science*, pages 215–226. Springer, 2004. Book: Mobile and Ubiquitous Information Access.

[Zim96] C. Zimmermann. *Advances in Object-Oriented Metalevel Architectures and Reflection.* CRC Press, Inc., Boca Raton, FL, USA, 1996.

[ZPG04] A.F. Zambrano, L. Polasek, and S. Gordillo. Desacoplando la personalización en las aplicaciones móviles. In *Actas del V Congreso Español en Interacción 2004*. Universitat de Lleida, Spain, 2004. http://griho.udl.es/i2004/BajarPonencia/37.pdf.

[ZVG06] A.F. Zambrano, T. Vera, and S.E. Gordillo. Solving aspectual semantic conflicts in resource aware systems. In W. Cazzola, S. Chiba, Y. Coady, and G. Saake,

editors, *Proc. of 1rd. Workshop on Reflection, AOP, and Meta-Data for Software Evolution (RAM-SE'06-ECOOP'06)*, pages 79–88. Fakultät für Informatik, Universität Magdeburg, 2006.