UNIVERSITAT JAUME I
DPTO. DE LENGUAJES Y SISTEMAS INFORMÁTICOS

Reutilización de servicios web mediante componentes integrados

Tesis Doctoral
Presentada por Carlos Granell Canut
Dirigida por el Doctor D. Michael Gould Carlson

Castellón, julio de 2006

Reutilización de servicios web mediante componentes integrados

Carlos Granell Canut

Trabajo realizado bajo la dirección del Doctor
D. Michael Gould Carlson
y presentado en la Universitat Jaume I
para optar al grado de Doctor por la Universitat Jaume I

Castellón, julio de 2006

Este trabajo ha sido parcialmente subvencionado por una beca FPI de la Generalidad Valenciana, y por los proyectos de la Unión Europea IST 2001-37724 y SST4-2004-012257 y el proyecto TIC2000-1568-C03-02 del Ministerio de Ciencia y Tecnología.

A Tefi.

A mis padres, hermano y sobrinas.

Resumen

Desde los inicios de la era de la información la reutilización ha supuesto una práctica habitual para el desarrollo de aplicaciones software. Actualmente, la reutilización de software continúa siendo una aspecto esencial en los sistemas de información actuales por el uso de componentes software, vistos como colecciones de código reutilizables que facilitan el desarrollo de aplicaciones software.

La aparición de los servicios web ha proporcionado un modelo diferente para el desarrollo de aplicaciones, ofreciendo la capacidad de acceder a servicios heterogéneos de forma unificada e interoperable a través de Internet. Nuevas aplicaciones web son creadas componiendo o combinando servicios web heterogéneos. Desafortunadamente, los desarrollos actuales prestan escasa atención a la reutilización de composiciones de servicios, como mecanismo para facilitar y reducir la complejidad existente a medida que crece el número de servicios web involucrados en una composición.

La contribución más importante de esta tesis ha consistido en aportar una aproximación para la composición de servicios web con el fin de facilitar el desarrollo de aplicaciones web basadas en servicios flexibles y reutilizables. Por una parte, la aproximación propuesta introduce el concepto de *componente integrado*, visto como una unidad básica reutilizable, que integra las características más relevantes sobre reutilización presente en los sistemas basados en componentes software y en el área de los workflows. Por otra parte, esta aproximación describe una *metodología de composición* para la creación, composición, reutilización de componentes integrados para transformarlos finalmente en procesos ejecutables.

Palabras clave: reutilización de servicios web, composición servicios web, componentes integrados, composiciones abstractas o virtuales, patrones de workflow, procesos ejecutables, WSDL-S, WSBPEL.

II

Abstract

From the beginnings of the era of the information the reusability has supposed a habitual practice for the development of software applications. At the moment, software reuse keeps being an essential aspect in the current information systems by the use of software components, as collections of reusable code that facilitate the development of software applications.

The emergence of web services has provided a different model for the applications development, offering the capacity of acceding heterogeneous services in a unified and interoperable way through Internet. New web applications are created by composing or combining heterogeneous web services. Unfortunately, current developments pay little attention to the reusability of service compositions, as mechanism to facilitate and reduce the existing complexity as the number of web services involved in a composition grows.

The most important contribution of this thesis has consisted of presenting an approach for web service composition with the aim to facilitate the development of web applications based on flexible and reusable services. On the one hand, the proposed approach introduces the concept of *integrated component*, as the reusable basic unit, that integrates the reusability characteristics from the component-based systems and workflow area. On the other hand, this approach describes a *composition methodology* for the creation, composition, and reuse of integrated components to finally transform them into executable processes.

Keywords: web service reuse, web service composition, integrated components, abstract or virtual compositions, workflow patterns, executable processes, WSDL-S, WSBPEL.

Agradecimientos

El trabajo de esta tesis nunca se hubiera completado sin la ayuda de mucha gente, a quienes me gustaría expresar mi gratitud en estas pocas líneas.

En primer lugar agradecer a mi director de tesis Dr. Michael Gould quien ha confiado plenamente en mi trabajo. Su paciencia y conocimiento científico han sido decisivos en la realización de este trabajo de investigación.

También quiero agradecer a los miembros de mi grupo de investigación y demás compañeros del departamento de Lenguajes y Sistemas Informáticos de la Universitat Jaume I, ya que de un modo u otro han contribuido a crear un entorno de trabajo incomparable para poder finalizar este trabajo. También tengo que agradecer profundamente las aportaciones y críticas de otros investigadores que han intervenido en diferentes fases de esta tesis, como David Skogan y Roy Grønmo de SINTEF; Rob Lemmens, Andreas Wytzisk y Rolf de By del ITC; y el Prof. Yvan Bédard de la Université Laval. Gracias a todos.

Quiero agradecer el apoyo y empuje incondicional de todos mis amigos y amigas a lo largo de estos últimos años, con los que he compartido largas cenas y tertulias en *La Casa*. Sería imposible enumerarlos a todos ellos así que, gràcies!.

Finalmente, dedico esta tesis a mi familia. Sin su suporte, ayuda y sacrificio durante mi vida de estudiante no hubiera sido posible este reto. No quisiera terminar sin expresar mi más sincero agradecimiento a Tefi por su paciencia y confianza, además de ser una excelente documentalista.

Castellón, Abril 2006

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Este capítulo describe la motivación y los objetivos de esta tesis. El capítulo también presenta los proyectos que han marcado el desarrollo de esta tesis junto con la metodología llevada a cabo, para finalizar con la estructura de la tesis.

1.1. Motivación

Ya desde los inicios de la era de la información se propuso el desarrollo de aplicaciones software mediante la composición de módulos de componentes software reutilizables [92]. El concepto subyacente consistía simplemente en agrupar colecciones de código en módulos reutilizables, llamados componentes, potenciando de este modo el desarrollo de aplicaciones software mediante la encapsulación y reutilización.

La reutilización de software es y seguirá siendo un aspecto esencial en los sistemas de información actuales. Un reciente informe difundido por la Comisión Europea[1], en el cual se recogen las visiones de futuro de reconocidos expertos sobre el software y los servicios identificando además las posibles líneas de investigación para el séptimo Programa Marco de la Unión Europea, expone que

> *The future state of software and services will require to mask the complexity of integration across multiple domains. This challenge cannot be overestimated. Users of services will demand efficient streamlined easy-to-use composition of services.*

[1] `ftp://ftp.cordis.lu/pub/ist/docs/directorate_d/st-ds/fp7-report_en.pdf`

Partiendo de esta sentencia, esta tesis se basa en la premisa de que son necesarios mecanismos más sencillos que den soporte a un modelo de composición de servicios web más flexible y eficiente, capaz de ofrecer al usuario aplicaciones particulares y a medida según los requisitos demandados. La reutilización de software puede ser una técnica apropiada que ofrezca mecanismos para la composición de servicios web con mayor grado de encapsulación, haciendo más simple al usuario la tarea de integrar diferentes servicios web.

Un buen ejemplo lo proporcionan las Infraestructuras de Datos Espaciales (IDE), que describen y engloban un conjunto de políticas, acuerdos institucionales y económicos, datos espaciales, servicios geo-espaciales y tecnologías que facilitan la disponibilidad y el acceso a los servicios y a los datos geo-espaciales [104]. Un usuario puede acceder directamente a los datos y servicios que proporciona una IDE, como pueden ser los datos geo-espaciales disponibles a través de geo-portales [88]. Sin embargo, la composición de servicios permite la posibilidad de crear nuevos servicios compuestos *ad hoc* a partir de otros servicios más simples ya disponibles. Desde la perspectiva del usuario, resulta interesante (en términos económicos, ahorro de tiempo, recursos, etc.) la creación de nuevos servicios compuestos reutilizando datos y servicios geo-espaciales ya existentes en una IDE, en vez de crear cada vez dichos servicios compuestos desde cero, desaprovechando en consecuencia recursos y duplicando servicios con la misma funcionalidad. En este sentido, la reutilización se convierte en un aspecto crítico que facilita el desarrollo de aplicaciones basadas en los servicios de una IDE o de cualquier infraestructura de datos en general.

1.2. Objetivos

Esta tesis introduce un modelo para la composición de servicios web basado en componentes integrados. Una de las principales características de este modelo es el soporte y la especial atención a la reutilización como mecanismo para crear nuevas composiciones de servicios web. La reutilización se aplica tanto a composiciones de servicios web como a las partes o composiciones contenidas en una composición dada, con el fin de maximizar lo máxima posible la reutilización. Para permitir diferentes niveles de reutilización, se define el concepto de componente integrado como la unidad básica reutilizable del modelo propuesto. Por lo tanto, la adecuada definición de un componente integrado junto con otras cuestiones asociadas a este concepto que veremos a lo largo de esta tesis, ha sido uno de los principales problemas a resolver durante el desarrollo de esta tesis para intentar dar respuesta a las preguntas que se plantean a continuación:

- ¿Podemos reutilizar composiciones existentes y partes de éstas para crear nuevas composiciones de servicios web?.
- ¿Qué características son deseables y cómo influyen éstas en la especificación de composiciones de servicios web reutilizables?.

Responder a la primera pregunta implicará explorar la noción de reutilización del software a lo largo de la historia de la computación para intentar aplicarla al contexto de los servicios web. El modelo propuesto se basa en el concepto de componente integrado, que parte necesariamente del estudio de los componentes software y de los workflows, áreas que tradicionalmente utilizan y explotan la reutilización del software. La respuesta a la segunda pregunta proporcionará la definición del componente integrado para que sea una unidad reutilizable, flexible, independiente, con cierto grado de encapsulación y con interfaces bien definidas. La definición de componente integrado implicará el desarrollo de una metodología apropiada para la creación, composición y reutilización de componentes integrados.

Esta tesis no intenta crear un modelo de reutilización de servicios web entre diferentes contextos de dominio, donde un servicio web cualquiera puede ser sustituido por otro sin importar si un servicio pertenece al contexto de planificación urbana y otro permite obtener recetas culinarias. Nuestra intención es ofrecer un modelo evitando la problemática de la integración de ontologías y de información heterogénea [49], centrándonos para ello en un único contexto de dominio, generalmente en el ámbito de la información geográfica.

En resumen, esta tesis persigue los siguientes objetivos:

- La exploración de la noción clásica de reutilización del software para ser aplicada a los servicios web.
- El desarrollo de un modelo que facilite la reutilización de composiciones de servicios web como mecanismo de composición.
- La aplicación del modelo propuesto en escenarios concretos, como el de una IDE, para examinar sus ventajas y limitaciones.

1.3. Contexto

El trabajo desarrollado en esta tesis ha sido posible gracias a los siguientes proyectos:

- *Adaptable and Composable E-Commerce and Geographic Information Services*[2] (IST 2001-37724). El objetivo principal del proyecto ACEGIS fue proporcionar un conjunto de herramientas más eficientes para el desarrollo, despliegue, descubrimiento y composición de servicios web, con especial énfasis a los servicios de información geográfica y de comercio electrónico. Claramente, gran parte del trabajo de esta tesis se ha desarrollado en el marco de este proyecto.
- *A tool for monitoring and forecasting Available WAter REsources in mountain environments*[3] (SST4-2004-012257). La finalidad de este proyecto en

[2] `http://www.acegis.net`
[3] `http://www.aware-eu.info`

curso es el desarrollo de herramientas innovadores para el seguimiento y la detección del agua disponible así como su distribución en las cuencas hidrográficas, donde el agua de deshielo es una componente importante en el balance hídrico anual, como por ejemplo en las cuencas alpinas. Todos los modelos hídricos del proyecto serán implementados mediante un conjunto de geo-servicios interactivos que permitirán a los usuarios del proyecto aplicar los modelos hídricos a cuencas alpinas tanto con datos locales como remotos.

- *Desarrollo de servicios distribuidos de catálogo de información geográfica orientada a Internet y basada en estándares abiertos. Pasos efectivos hacia una Infraestructura Nacional de Información Geográfica*[4] (referencia TIC2000-1568-C03-02), subvencionado por el Ministerio de Ciencia y Tecnología.

1.4. Metodología

La metodología que hemos adoptado para alcanzar los objetivos propuestos se puede resumir en los siguientes puntos:

- La investigación y análisis del estado del arte en la composición y reutilización de servicio web.
- El estudio de la noción de reutilización en el contexto de los componentes software para posteriormente ser aplicada al contexto de los servicios web.
- El estudio y análisis de los patrones de workflow en su faceta más tradicional en el área de los workflows, para ser aplicados posteriormente al contexto de la composición de servicios web.
- Introducción del concepto de componente integrado como pieza clave para la reutilización, capturando las características más interesantes, desde el punto de vista de la reutilización, de los componentes software y de los patrones de workflow.
- Introducción de una metodología para el diseño de componentes integrados junto con una arquitectura conceptual para darle soporte.
- Implementación de una herramienta software para el diseño de componentes integrado y evaluación de nuestra aproximación aplicando el modelo propuesto a un escenario concreto.

[4] `http://redgeomatica.rediris.es/metadatos/`

1.5. Estructura

Los restantes capítulos de esta tesis se organizan de la siguiente manera:

El capítulo 2 introduce conceptos claves para el desarrollo de nuestro modelo basado en componentes integrados, sobre todo en el área de los componentes software, los workflows y los servicios web. El capítulo también realiza un estado del arte sobre las técnicas y metodologías actuales en la composición y reutilización de servicios web.

El capítulo 3 introduce el concepto de componente integrado, pieza básica de nuestro modelo para la reutilización de composiciones de servicios web. Este capítulo también muestra tanto las características de los servicios web, componentes software y patrones de workflow que son interesantes para nuestro modelo, así como la influencia que ejercen en la definición de un componente integrado.

El capítulo 4 completa nuestro modelo detallando una metodología de composición para la creación, composición y transformación de componentes integrados.

El capítulo 5 muestra la herramienta software para el diseño de componentes integrados desarrollada en esta tesis aplicada a un escenario de control de emergencias, ilustrando las características más destacables de nuestro modelo basado en componentes integrados. Este capítulo también analiza los resultados obtenidos mediante un estudio comparativo en términos cualitativos.

El capítulo 6 resume los resultados de investigación obtenidos, presenta las conclusiones y futuras direcciones de investigación, así como las publicaciones más representativas derivadas del trabajo realizado en esta tesis.

Capítulo 2

Estado del arte

Este capítulo introduce conceptos claves para el desarrollo de nuestro modelo basado en componentes integrados, sobre todo en el área de los componentes software, los workflows y los servicios web. El capítulo también presenta un estudio del estado del arte en la composición y reutilización de servicios web.

2.1. Introducción

Esta tesis trata sobre la reutilización de servicios web a partir de conceptos provenientes de tres áreas distintas. Nuestro trabajo relaciona los componentes software y los patrones de workflow en el contexto de los servicios web. Como veremos a lo largo de este capítulo, todas estas áreas de investigación siguen teniendo influencia en la creación de aplicaciones y sistemas a partir de componentes reutilizables que pueden ser compuestos [24].

Las secciones de este capítulo se clasifican en dos partes bien diferenciadas. La primera parte (secciones 2.2, 2.3 y 2.4) muestra las tecnologías y conceptos base de esta tesis. El orden de exposición sigue la evolución histórica de estas tecnologías, partiendo desde los componentes software (sección 2.2), luego pasando a los workflows y los patrones de workflow (sección 2.3), para finalizar revisando la arquitectura y tecnología relacionada con los servicios web (sección 2.4).

La segunda parte (secciones 2.5 y 2.6) realiza un estudio del estado del arte sobre la composición y reutilización de servicios web. Hemos preferido crear dos secciones separadas para focalizar mejor el problema, y en consecuencia la contribución de la tesis, yendo desde el problema general (composición de servicios web) a un problema específico de la composición (reutilización).

2.2. Componentes software

Ya desde los inicios de la era de la información, se propuso el desarrollo de aplicaciones software mediante la composición de módulos de componentes software reutilizables [92]. La idea clave era agrupar colecciones de código en módulos reutilizables, llamados componentes, impulsando de este modo la encapsulación y la reutilización en el desarrollo de aplicaciones software. En esta dirección, a principios de la década de los noventa, el término *megaprogramming* fue propuesto en [160] para describir la creación de aplicaciones software mediante la composición de *megamodules*, haciendo indistinguible la utilización o reutilización de dichos *megamodules*.

En la actualidad, el principio básico de los sistemas de software basados en componentes se centra en la construcción de aplicaciones mediante la composición de componentes software reutilizables [24, 142]. Ejemplos destacables pueden ser OMG CORBA [153], Microsoft DCOM [37], y los SUN Enterprise Java Beans (EJB) [115]. Un componente software es una pieza de software reutilizable, independiente y con una interfaz bien definida, capaz de componerse con otros componentes en diferentes entornos. Desde la perspectiva de esta tesis, es interesante recalcar las siguientes características que definen a un componente software:

- *Reutilización.* Un mismo componente puede formar parte de diferentes aplicaciones software en diferentes contextos. La reutilización es la característica más destacada de este tipo de sistemas.
- *Encapsulación.* Desde la perspectiva de una aplicación (u otro componente), un componente debe entenderse como un caja negra cuya funcionalidad interna es únicamente accesible a través de una interfaz bien definida. De esta forma, la comunicación entre componentes se realiza a través de dichas interfaces.
- *Independencia.* Un componente es independiente porque es capaz de realizar su función de forma autónoma, es decir, no depende de otros componentes para llevar a cabo su tarea.

Los componentes software juegan un papel primordial en esta tesis puesto que las anteriores características también estarán presentes en el concepto de componente integrado (véase capítulo 3), concepto esencial en nuestro modelo para la reutilización de servicios web, visto como una unidad básica reutilizable, independiente, y con la funcionalidad encapsulada mediante interfaces bien definidas.

2.3. Workflows

La idea de workflow o proceso de negocio se remonta a finales de los setenta, pero no es hasta la década de los noventa cuando los Workflow Management

Systems (WfMS) llegan a consolidarse como un elemento esencial en los sistemas de información empresariales [149]. Es fácil imaginar un WfMS como un sistema operativo que controla, administra, y ejecuta workflows, siendo un workflow una colección de actividades o tareas relacionadas entre sí mediante flujos de datos y de control [94]. Un WfMS contiene, entre otros elementos, un motor de workflow que permite interpretar y ejecutar una determinada instancia de un workflow, generalmente de un modo centralizado aunque también existen recientes estudios sobre la ejecución descentralizada de workflows [152].

El desarrollo de las redes de comunicaciones permitió que las organizaciones se interconectasen con mayor facilidad, haciendo que los workflows internos de las organizaciones transpasasen los límites de la propia organización para interactuar con otras organizaciones. Los WfMS pasaron a convertirse en una tecnología clave para la integración de workflows entre diferentes organizaciones en un contexto B2B (*Business-to-Business*) [27]. A su vez, este nuevo escenario B2B requirió de nuevos lenguajes para la especificación de workflows, principalmente promovidos por el Workflow Management Coalition (WfMC[1]), con el fin de declarar, controlar y monitorizar workflows mediante protocolos basados en Internet [66]. Sin embargo, estos lenguajes estaban centrados más en la integración de workflows en vez de promover la interacción de workflows en un entorno B2B. Por este motivo surgió Electronic Business XML (ebXML[2]) [120], con el propósito de definir un conjunto de especificaciones que permitiesen las interacciones B2B entre empresas y compañías de cualquier naturaleza y tamaño, con el fin último de alcanzar un mercado electrónico global.

El auge de los WfMS ha permitido que prácticamente cada WfMS comercial implemente su propio lenguaje para la especificación de workflows, a pesar de los esfuerzos de estandarización del WfMC [66]. La falta de uniformidad entre los diferentes WfMS llevó a van der Aalst et al. [148] a plantear un conjunto uniforme de patrones de workflow con el que modelar cualquier flujo de control de actividades en un workflow. Este conjunto de patrones fue ampliamente aceptado por los principales vendedores de WfMS [148], lo que contribuyó a convertirlos en indicadores fiables y objetivos con los que comparar la funcionalidad de los diferentes WfMS existentes en el mercado. Por este motivo, en esta tesis nos hemos basado en dichos patrones para desarrollar un conjunto propio de patrones de composición para los componentes integrados (véase capítulo 3). Es necesario, pues, realizar una descripción básica de cada uno de los patrones de workflow propuestos por [148] para facilitar la lectura y comprensión de capítulos posteriores de esta tesis.

2.3.1. Patrones de workflow

La Tabla 2.1 muestra los 20 patrones [148] agrupados en seis categorías según su funcionalidad. Algunos patrones son obvios mientras que otros re-

[1] `http://www.wfmc.org`
[2] `http://www.ebxml.org`

sultan un poco más extraños e inusuales. De cualquier modo, el lector interesado puede encontrar en [76, 148, 65] una descripción exhaustiva de cada uno de estos patrones, así como una demostración animada de cada patrón en `http://www.workflowpatterns.com`, facilitada por sus propios autores.

Basic control flow patterns

La primera categoría se denomina *basic control flow patterns* y engloba los patrones que capturan las capacidades fundamentales presentes en cualquier workflow como la secuencia, paralelismo o la sincronización de varias actividades concurrentes.

- *Sequence* (1). Este patrón ejecuta actividades en secuencia.
- *Parallel split* (2). El patrón *parallel split* se utiliza para la ejecución de varias actividades en paralelo sin ninguna sincronización entre ellas. Por ejemplo, después de la actividad A se ejecuta al mismo tiempo tanto la actividad B y C.
- *Synchronization* (3). El patrón *synchronization* permite que varias bifurcaciones en paralelo converjan hacia una única actividad A, esperando a que todas las actividades concurrentes terminen antes de comenzar con dicha actividad A.
- *Exclusive choice* (4). Dadas varias actividades, este patrón activa únicamente una de éstas según se cumpla cierta condición.
- *Simple merge* (5). Este patrón es el complementario del anterior, realizando únicamente la sincronización de la actividad iniciada por el patrón *exclusive choice*.

Advanced branching and synchronization patterns

En contraposición a los anteriores patrones, la categoría *advanced branching and synchronization patterns* incluye patrones mucho más avanzados que los anteriores pero igualmente importantes en cualquier workflow.

- *Multi-choice* (6). Dadas varias actividades concurrentes, este patrón activa una o varias de éstas según se cumpla cierta condición.
- *Synchronization merge* (7). Este patrón es el complementario del anterior, esperando a que todas las actividades concurrentes iniciadas por el patrón *multi-choice* se completen.
- *Multi-merge* (8). El patrón *multi-merge* permite que a medida que las actividades concurrentes terminan, se crea un hilo de ejecución independiente para cada una de éstas para que continúe la ejecución del workflow sin esperar a que se completen todas las demás actividades. En este patrón convergen todas las actividades pero sin sincronización.

Id.	Patrón de workflow	Descripción
Basic control flow patterns		
1	Sequence	Secuencia de actividades
2	Parallel split	Activa todas las actividades concurrentes de las posibles
3	Synchronization	Sincroniza actividades iniciadas por 2
4	Exclusive choice	Activa solo una de varias actividades concurrentes
5	Simple merge	Sincroniza actividad iniciada por 4
Advanced branching and synchronization patterns		
6	Multi-choice	Activa una o varias actividades concurrentes de las posibles
7	Synchronizing merge	Sincroniza actividades iniciadas por 6
8	Multi-merge	Actividad concurrente termina sin sincr.
9	Discriminator	Solo una de varias actividades concurrentes continúa
Structural patterns		
10	Arbitrary cycles	Repite una a varias actividades
11	Implicit termination	Termina ejecución de un proceso (ej. bucle) para reducir complejidad
Patterns involving multiple instances		
12	M.I.without synchronization	Activas múltiples instancias sin sincr.
13	M.I.with design time knowledge	Activas n instancias con sincr., n conocido en tiempo diseño
14	M.I.with runtime knowledge	Activas n instancias con sincr., n conocido en tiempo ejecución
15	M.I.without runtime knowledge	Activas múltiples instancias (número desconocido) con sincr.
State-based Patterns		
16	Deferred choice	Activa una de varias actividades concurrentes según evento externo
17	Interleaved parallel routing	Secuencia de actividades con orden arbitrario
18	Milestone	Una actividad puede ejecutarse durante el intervalo de tiempo entre la activación y la expiración de un *milestone*
Cancellation patterns		
19	Cancel activity	Aborta una actividad
20	Cancel case	Aborta workflow entero

Tabla 2.1: Patrones de workflow

- *Discriminator* (9). El patrón *discriminator* es interesante porque permite que varias actividades concurrentes converjan a una actividad A, pero únicamente el flujo de ejecución de una de estas actividades prosigue con la actividad A, según se cumpla cierta condición evaluada en tiempo de ejecución, mientras que los demás hilos de ejecución de las otras actividades son ignorados.

Structural patterns

Los patrones estructurales, como su nombre implica, permiten el diseño de un flujo de actividades más organizado, facilitando la comprensión del workflow.

- *Arbitrary cycles* (10). Básicamente, este patrón repite una o varias actividades pero de forma similar a los saltos GOTO de los lenguajes de programación. Este patrón difiere de los bucles WHILE porque permite que existan varios puntos de entrada y salida, aspecto necesario para la creación de workflows en ciertos escenarios reales.

- *Implicit termination* (11). El patrón *implicit termination* termina la ejecución de cierto subproceso de un workflow (por ejemplo bucles o subrutinas) simplemente porque no hay nada más que hacer. Tiene sentido cuando mediante este patrón se reduce la complejidad de un subproceso, similar por ejemplo a las instrucciones RETURN que terminan la ejecución en cualquier punto de un bucle o subrutina, que de otro modo resultaría mucho más complejo o menos intuitivo.

Patterns involving multiple instances

En todos estos patrones una actividad puede tener más de una instancia activa y ejecutándose al mismo tiempo. El patrón *without synchronization* (12) no implica sincronización mientras que en los tres restantes (13-15) existe sincronización entre las múltiples instancias concurrentes de una actividad.

- *Without synchronization* (12). Mediante este patrón, múltiples instancias de una actividad se ejecutan concurrentemente pero sin que exista una sincronización entre las distintas instancias a medida que se completan.

- *With design time knowledge* (13). Esta vez, el número de instancias concurrentes de una actividad se conoce de antemano en tiempo de diseño. Además existe sincronización entre todas las instancias para que se completen antes de continuar con el resto de actividades del workflow.

- *With runtime knowledge* (14). Idéntico al patrón anterior, salvo que el número de instancias concurrentes es determinado en tiempo de ejecución.

- *Without runtime knowledge* (15). Similar al patrón *without synchronization* (13) ya que el número de instancias concurrentes es desconocido

hasta cierto momento durante el procesamiento de las instancias. Sin embargo, todas las instancias activas deben sincronizarse para continuar con el resto de actividades del workflow.

State-based patterns

Típicamente, las actividades en escenarios de workflow tradicionales permanecen más tiempo en estado de espera que en proceso [76]. Esto significa que las actividades son activadas cuando reciben ciertos mensajes de activación mediante un sistema de notificación de eventos. Los siguientes patrones describen situaciones en que el estado de una actividad cambia en función de un evento externo a la propia actividad.

- *Deferred choice* (16). Es prácticamente idéntico al patrón *exclusive choice* (4) exceptuando la naturaleza de la condición. En ambos patrones una de las posibles actividades es activada dependiendo de una condición. En el patrón *exclusive choice* la condición viene dada por decisiones o datos de la propia actividad, mientras que en el patrón *deferred choice* la condición se basa en decisiones sobre datos externos notificadas mediante eventos.
- *Interleaved parallel routing* (17). Define la ejecución de una secuencia de actividades sin ningún orden establecido en tiempo de diseño. Por ejemplo dadas las actividades A, B y C, el patrón *interleaved parallel routing* ejecuta arbitrariamente tales actividades en secuencia, dando lugar a una de las siguientes combinaciones: ABC, ACB, BAC, BCA, CAB, CBC.
- *Milestone* (18). Se trata de un patrón bastante inusual y diferente al resto de patrones. El patrón *milestone* describe un posible escenario en el que una actividad puede ser ejecutada durante un intervalo de tiempo marcado por la activación y terminación de cierto "milestone" o evento. El ejemplo de una subasta propuesto en [65] esclarece el funcionamiento de este patrón: la actividad "puja" puede llevarse a cabo en cualquier momento desde el principio y final del tiempo de puja ("milestone" o evento), una vez finalizado el tiempo de puja ya no es posible realizar ninguna actividad "puja".

Cancellation patterns

Los patrones de cancelación permiten abortar la ejecución de un actividad o incluso de todo un workflow en cualquier punto de su ejecución.

- *Cancel activity* (19). Permite la cancelación de la ejecución de una actividad para abortar, por ejemplo, una actividad suspendida o de larga duración, prosiguiendo la ejecución del proceso con la siguiente actividad. Un ejemplo sencillo de este patrón es la cancelación de una actividad de impresión de un trabajo porque se ha producido un determinado error en la propia impresora.

- *Cancel case* (20). Mediante este patrón se aborta completamente la ejecución de un workflow.

2.4. Servicios web

Existen numerosas definiciones del término *servicio web* en la literatura. Una de las definiciones más utilizada comúnmente es la proporciona por Alonso et al. [7] que define un servicio web como

> *A business function made available via the Internet by a service provider, and accessible by clients that could be human users or software applications.*

Otros autores proponen que los servicios web pueden ser vistos como [94]

> *Loosely coupled applications using open, cross-platform standards and which interoperate across organizational and trust boundaries.*

Por otra parte, el organismo World Wide Web Consortium (W3C[3]) define un servicio web como

> *A software application identified by a URI (Uniform Resource Identifier), whose interfaces and binding are capable of being defined, described and discovered by XML artifacts and supports direct interactions with other software applications using XML-based messages via Internet-based protocols.*

Las definiciones anteriores proporcionan visiones complementarias acerca de lo que es un servicio web. Básicamente, un servicio web describe cierta funcionalidad accesible a través de Internet mediante estándares independientes de la plataforma. Con esta definición queremos enfatizar ciertas características clave de un servicio web. En primer lugar, los servicios web están débilmente acoplados (*loosely coupled*) debido principalmente a que la descripción funcional está separada de la implementación concreta del servicio. Los servicios web también proporcionan interoperabilidad, ya que cualquier servicio web puede interaccionar con otro mediante un modelo de comunicación basado en el intercambio de documentos XML, siendo independiente de la plataforma y del lenguaje utilizado [146]. Finalmente, resulta relativamente rápido y fácil el desarrollo de servicios web encima o "envolviendo" sistemas de información ya existentes (*legacy systems*), evitando de este modo los desarrollos completos de aplicaciones software normalmente mucho más costosos en tiempo, dinero y recursos humanos.

A primera vista, los servicios web suelen confundirse con los sistemas basados en componentes [142] y la tecnología orientada a objetos [144]. Los servicios

[3] `http://www.w3c.org`

web son tecnologías para sistemas distribuidos pero no son objetos distribuidos [155], ya que no comparten ninguna de las características que definen a un objeto, empezando por la propia noción de objeto, referencias a objetos o el ciclo de vida de un objeto. Los servicios web resultan más afines a los sistemas basados en componentes ya que ambos tienen mecanismos similares para publicar y localizar sus componentes así como un lenguaje para describir las interfaces. Sin embargo, otras características diferencian claramente los servicios web de los componentes, tal como veremos en la sección 3.2.1 del siguiente capítulo.

2.4.1. Modelo referencia

El modelo de referencia de los servicios web puede considerarse como una versión minimalista de la arquitectura orienta a servicios (Service-Oriented Architecture, SOA) [69]. Una arquitectura SOA está compuesta exclusívamente de servicios, que pueden ser dinámicamente localizados y compuestos según las funcionalidades expresadas en sus descripciones. Una parte integral de SOA son los servicios que cumplen dos características básicas: las descripciones deben estar bien definidas para que sean independientes de la implementación del servicio y además, los servicios deben ser independientes de los demás, es decir, su ejecución no involucra la ejecución de otros servicios. No debe entenderse SOA como una implementación concreta sino como una arquitectura conceptual definida por un conjunto de normas, capas de abstracción, procedimientos y políticas que nos acercan al 'estilo' SOA. En definitiva, la tecnología de los servicios web es tan solo una forma de materializar una arquitectura SOA.

Ya que últimamente han proliferado enormemente el número de especificaciones y tecnologías relacionas con SOA y los servicios web [154], hemos preferido describir el modelo de referencia de servicios web desde dos perspectivas diferentes pero complementarias al mismo tiempo: una visión funcional, donde describimos los roles y las funciones típicas de los servicios web, y una visión tecnológica, donde describiremos las diferentes especificaciones y tecnologías básicas utilizadas en el ámbito de los servicios web.

Visión funcional: roles y funciones

La Figura 2.1 ilustra las interacciones que existen entre los tres tipos de roles (o participantes) en el modelo de referencia funcional para los servicios web así como las funciones de cada uno de estos roles. Comencemos examinando cada uno de los roles:

- *Proveedor de servicios (service provider).* El rol de proveedor de servicios proporciona aplicaciones software como servicios web, creando descripciones de los servicios para hacerlas públicas en uno o varios registros de servicios y actualizando además los servicios (a nivel de implementación) cuando sea necesario. Un proveedor de servicios es el propietario de

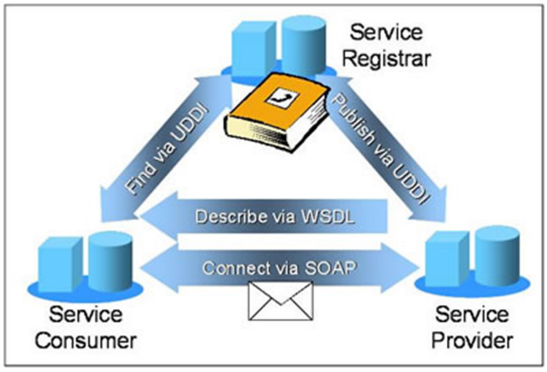


Figura 2.1: Modelo de referencia funcional para servicios web

los servicios web y generalmente, también actúa de plataforma donde se albergan las implementaciones concretas de estos servicios web.

- *Cliente de servicios (service consumer).* El rol de cliente de servicios desempeña al participante que precisa de ciertos requisitos que pueden ser cubiertos mediante uno o varios servicios web disponibles en Internet. Un cliente puede ser un usuario accediendo a un servicio web mediante un navegador; también puede ser una aplicación software, un agente [68] u otro servicio web. De hecho, es posible que un mismo servicio web desempeñe más de un rol, como por ejemplo que sea proveedor y cliente de servicios a la vez, como sucede con las composiciones de servicios web que se comportan como clientes respecto a los propios servicios web que contienen y como proveedores respecto al cliente que invoca la composición.

- *Registro de servicios (service register).* El rol de registro de servicios proporciona un repositorio de descripciones de servicios web publicadas por el proveedor de servicios. También proporciona los mecanismos de búsqueda en el repositorio que facilitan a los clientes de servicios la localización de servicios web apropiados a sus necesidades. Algunos ejemplos de repositorios de servicios web son BindingPoint[4] y XMethods[5].

Para complementar la visión funcional del modelo de referencia de servicios web, se definen las tres funciones que describen las interacciones entre los roles descritos anteriormente (véase Figura 2.1):

- *Publicar (publish).* La función publicar permite que un proveedor de servicios publique las descripciones de los servicios web en uno o más registros para hacer accesibles los servicios web a los clientes de servicios. Relaciona los roles proveedor y registro de servicios.

[4] `http://www.bindingpoint.com`
[5] `http://www.xmethods.com`

- *Buscar (find).* La función buscar permite que un cliente de servicios especifique ciertos criterios de búsqueda para recuperar las correspondientes descripciones de servicios web que se encuentran en el registro de servicios. Permite que los roles de cliente y registro de servicios interactúen.
- *Enlazar (bind).* Una vez el cliente de servicios localiza la descripción requerida, la función enlazar permite que dicho cliente de servicios acceda y se comunique con el respectivo proveedor de servicios, para invocar alguna de las operaciones detalladas en la descripción del servicio web.

Es evidente que para que exista una interacción entre el proveedor, el cliente y el registro de servicios, ésta debería plantearse en términos de interoperabilidad mediante el uso de especificaciones y tecnologías estándares, justo la visión tecnológica de los servicios web que plantea el siguiente punto.

Visión tecnológica: capas y tecnologías

La Figura 2.2, modificación de la propuesta en [40], representa el conjunto de capas, especificaciones y tecnologías en el mundo de los servicios web conocido como "pila de los servicios web". En realidad, no existe una visión unificada de esta pila de los servicios web, aunque las propuestas del W3C Web Services Architecture Working Group[6] y de la Organization for the Advancent of Structured Information Standards (OASIS[7]) tienen algunos elementos en común, también presentan claras divergencias debido al uso de especificaciones propias de cada organismo. A continuación nos centramos en la funcionalidad de cada una de estas capas en vez de proporcionar una lista exhaustiva de las especificaciones que contienen. Sin embargo, esbozaremos algunas características de las especificaciones básicas y que han sido utilizadas durante el desarrollo de esta tesis.

Transport. La capa *Transport* incluye tecnologías y protocolos que no son propios de los servicios web, pero representan los protocolos de transporte que dan soporte a las demás capas. La interoperabilidad, al menos a nivel de infraestructura, está garantizada con el uso de Internet como infraestructura común para el transporte y comunicación entre servicios web [133, 7]. De esta forma, los servicios web son independientes del protocolo de transporte utilizado (HTTP, SMTP, FTP, etc.) ya que es justo la responsabilidad de esta capa determinar cómo se transportan mensajes entre servicios web.

Base Technologies. Prácticamente, la totalidad de las tecnologías de servicios web descansan sobre el meta-lenguaje XML[8], que facilita la definición de lenguajes específicos para el intercambio de datos en el contexto Web.

[6] `http://www.w3.org/2002/ws/arch/`
[7] `http://www.oasis-open.org`
[8] `http://www.w3.org/XML`

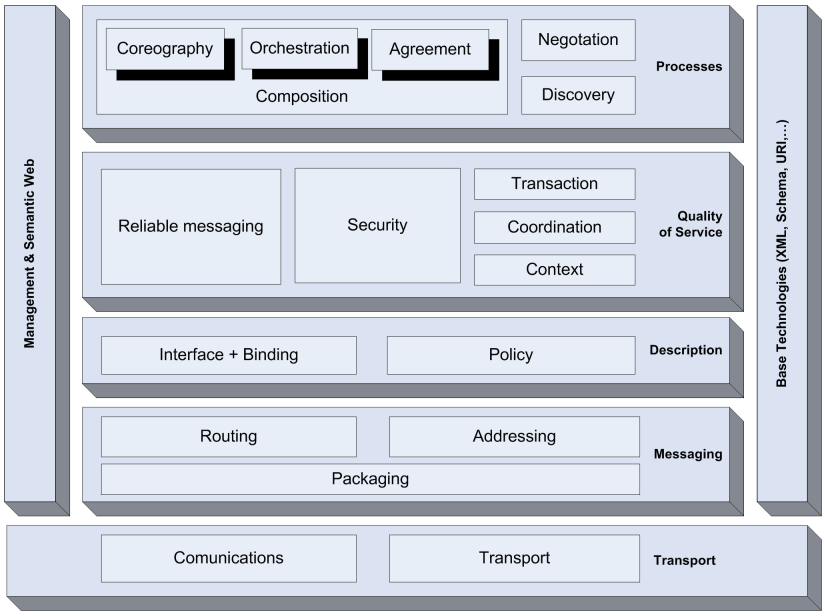


Figura 2.2: Pila de los servicios web

La adopción de XML como tecnología base resuelve el problema de la interoperabilidad a nivel sintáctico [133], ya que proporciona un formato común para expresar datos y contenido (documentos) compartidos entre aplicaciones [3, 132]. Además, una de sus principales ventajas radica en que el contenido de un documento XML puede ser validado a partir de su esquema, descrito mediante XML Schema[9]. Un ejemplo de este tipo de esquemas puede encontrarse en el Anexo A.

Messaging. Esta capa proporciona un modelo para "empaquetar" o codificar mensajes entre servicios web listos para ser transportados. Simple Object Access Protocol (SOAP[10]) es uno de los protocolos básicos de los servicios web [41, 146]. SOAP permite el intercambio de mensajes entre servicios web formateados en XML utilizando gran variedad de protocolos de transporte. Un mensaje SOAP puede imaginarse como un sobre (*envelope*) que envuelve básicamente dos elementos: un *header*, donde los protocolos de la capa *Quality of Service* pueden ser incluidos; y un *body* que contiene realmente los datos intercambiados entre servicios web.

Description. Una de las premisas de SOA es proporcionar servicios débilmente acoplados (*loosely coupled*) entre aplicaciones [69]. Esto implica

[9] `http://www.w3.org/XML/Schema`
[10] `http://www.w3.org/TR/soap`

que los servicios deben describirse adecuadamente mediante descripciones funcionales y no funcionales. La capa *Description* trata con los aspectos funcionales de un servicio mientras que la capa *Quality of Service* hace referencia a los aspectos no funcionales.

La especificación Web Service Description Language (WSDL[11]) es un lenguaje basado en XML que permite la descripción de interfaces funcionales de servicios web. WSDL incluye información acerca de las interfaces y de la implementación de dichas interfaces, proporcionando los protocolos específicos de transporte y de empaquetado para el acceso a un servicio web (*binding*). Por otro lado, las propiedades no funcionales pueden ser embebidas en la descripción de un servicio mediante otras especificaciones adicionales como WS-Policy[12], que permite codificar aspectos de seguridad, integridad, propiedades transaccionales y demás información propia de la capa *Quality of Service*.

Quality of Service. Esta capa incluye todos los aspectos no funcionales relacionados con un servicio web, como la seguridad, privacidad, propiedades transaccionales, disponibilidad o fiabilidad [97], que son de vital importancia para que la tecnología de los servicios web despegue definitivamente en el mundo del comercio electrónico (*e-commerce*) [1]. Multitud de especificaciones abarcan cada una de estas características aunque únicamente WS-Security[13] ha alcanza el estatus de especificación estándar [40].

Processes. La capa *Processes* incluye la composición y localización de servicios web. Una de las características más importantes de los servicios web es la capacidad de crear nuevas funcionalidades mediante la composición de servicios web. Ya que la composición es el foco de esta tesis, será tratada con mayor atención en la sección 2.5.

No menos importante en SOA es la capacidad de localizar dinámicamente servicios web, para enlazarlos e interaccionar con ellos. La especificación Universal Description, Discovery, and Integration (UDDI[14]) [41, 146], bajo el auspicio de OASIS, define una interfaz programable separada para la publicación y la búsqueda de servicios web. El principal elemento de UDDI es el registro de servicios basados en XML, a modo de listín telefónico denominado *business registry*. Básicamente, este registro contiene tres tipos de información para cada servicio web: la información de contacto (*white pages*), la categorización empresarial (*yellow pages*), y la información técnica (*green pages*) que contiene un enlace a la descripción WSDL del servicio web.

Normalmente, las interacciones y conversaciones entre servicios web son

[11] `http://www.w3.org/TR/wsdl`
[12] `http://www.ibm.com/developerworks/library/ws-polfram/`
[13] `http://www.ibm.com/developerworks/library/ws-secure/`
[14] `http://www.uddi.org`

reguladas mediante contratos [99, 15], en los cuales se especifican acuerdos entre las partes implicadas, el proveedor y el cliente de servicios. La capacidad de gestionar y negociar de forma dinámica estos acuerdos será un aspecto crítico para el futuro de los servicios web [40].

Management and Semantic Web. Como ilustra la Figura 2.2, la capa *Management and Semantic Web* es transversal a las restantes capas de los servicios web. La gestión de servicios web ganará atención a medida que aumente la cantidad de servicios web disponibles en Internet [33]. En este sentido, será cada vez más crítico disponer de herramientas capaces de monitorizar y controlar la ejecución de servicios web para, por ejemplo, tarifar el uso de servicios web en un contexto de comercio electrónico.

La web semántica, en palabras de sus visionarios [23], propone una extensión de la actual web donde la información tiene un significado bien definido, permitiendo de esta forma una mejor cooperación entre máquinas y personas. La idea subyacente es la asociación de semántica al contenido [134], es decir, la estructuración conceptual del contenido de la Web de un modo explícito para las máquinas. Las ontologías se han convertido en elementos esenciales de la web semántica, como mecanismos para describir y estructurar explícitamente el conocimiento en un dominio concreto, facilitando de este modo la interoperabilidad semántica entre aplicaciones [159]. La representación de una ontología se lleva a cabo mediante lenguajes ontológicos [52]. Uno de los primeros lenguajes de la era Web fue DAML+OIL [90, 38]. Posteriormente, como evolución de éste, apareció el lenguaje OWL Web Ontology Language[15], convirtiéndose finalmente en estándar para la representación de conocimiento en la Web por el W3C. El lenguaje OWL está formado por diferentes elementos: conceptos, relaciones, funciones, axiomas e instancias [52]. Sin embargo, en el contexto de esta tesis, únicamente nos fijaremos en los conceptos (atributos `class` y `property` en OWL) ya que representan la jerarquía de conceptos de un dominio concreto (a modo de taxonomías), utilizados para la anotación semántica de las descripciones de servicios web.

La llegada de la web semántica a los servicios web produjo el nacimiento de los servicios web semánticos [91, 118], aplicando las características de los lenguajes ontológicos a la representación de los servicios web para permitir la localización, composición y ejecución automática de servicios web semánticos. En general, existen dos aproximaciones para añadir semántica a los servicios web [30]. La primera consiste en la utilización de una ontología para describir las capacidades y requerimientos de los servicios web. Ejemplos de este tipo de ontologías son OWL-S[16] y Web Service Modeling Framework (WSMF) [47] que posteriormente derivó en el Web

[15] `http://www.w3.org/TR/owl-ref/`
[16] `http://www.w3.org/Submission/OWL-S`

Service Modeling Ontology (WSMO[17]) [45, 44]. Un estudio comparativo entre ambas ontologías para servicios web semánticos puede encontrarse en [82]. Debido a la complejidad para describir y componer servicios web basados en ontologías de servicios web, la anotación de servicios web [4] surge como un mecanismo sencillo y rápido para añadir semántica aprovechando los estándares básicos de los servicios web (principalmente WSDL y UDDI). A partir de conceptos definidos en ontologías compartidas, WSDL-S [138] proporciona un mecanismo para enriquecer semánticamente las descripciones de las operaciones y parámetros definidos en WSDL, utilizando para ello etiquetas semánticas que aprovechan la característica de extensibilidad de la especificació WSDL. En esta tesis hemos utilizado WSDL-S como mecanismo para la anotación semántica de las descripciones de los componentes integrados.

Aunque en esta tesis tratamos con ciertos aspectos semánticos, en realidad no es nuestra intención abordar el problema de la composición de servicios web desde un punto de vista semántico. Sin embargo, vale la pena mencionar como el estudio de los servicios web semánticos se aplica de forma transversal a las diferentes capas de los servicios web, tal como muestra la Figura 2.2. Así, existen trabajos para la definición de una arquitectura conceptual de servicios web semánticos [29], la localización de servicios web semánticos [73, 72, 137, 151], la composición automática de servicios web semánticos [28, 31, 35, 53, 63, 83, 95, 103, 136, 141, 145], sobre descripciones formales para servicios web semánticos [81, 123] así como frameworks para el despliegue y la ejecución de servicios web semánticos [53, 117].

2.4.2. Servicios geo-espaciales

Tras haber introducido el concepto y el modelo de referencia de los servicios web, esta sección muestra algunos ejemplos de servicios en el contexto geo-espacial para complementar la visión técnica de la sección anterior, al mismo tiempo que introduce algunos servicios geo-espaciales que serán utilizados en el escenario planteado en el capítulo 5.

El Open Geospatial Consortium (OGC[18]) ha sido la organización más comprometida en proporcionar un contexto web a los datos geo-espaciales, guiando múltiples grupos de trabajo formados por profesionales e investigadores en el campo de los sistemas de información geográficos. La finalidad última de los grupos de trabajo OGC es la generación de interfaces que fomenten la interoperabilidad entre datos y servicios geo-espaciales [25, 133]. Como primer paso, las primeras especificaciones OGC describieron interfaces basadas en Internet (peticiones HTTP GET/POST explícitas) para acceder y proporcionar datos geo-espaciales a través de un navegador web. Sin embargo, como el acceso

[17] `http://www.wsmo.org`
[18] `http://www.opengeospatial.org`

a estos servicios no se realiza mediante XML ni sigue un modelo publicar-buscar-enlazar (véase sección 2.4.1), no puede considerarse que estos servicios se ajusten a la arquitectura de los servicios web.

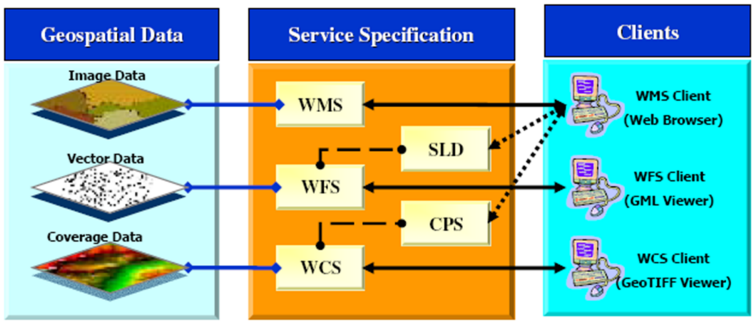


Figura 2.3: Especificaciones OGC básicas

La Figura 2.3 [84] muestra tres de las especificaciones para servicios geo-espaciales básicas junto con los tipos de datos geo-espaciales que sirven. La especificación Web Map Service (WMS) [113] transmite al cliente una vista o copia (generalmente una imagen PNG, GIF o JPEG) como resultado del procesamiento de mapas georeferenciados. Web Feature Service (WFS) [112] proporciona interfaces para filtrar y recuperar representaciones vectoriales de características geo-espaciales (*features*) como puntos, líneas o polígonos, codificadas como documentos GML (Geography Markup Language) [109]. El siguiente servicio geo-espacial básico, Web Coverage Service (WCS) [108], proporciona coberturas o datos raster (imágenes satélite) en vez de datos vectoriales como en el WFS. Cuando el cliente dispone de un navegador con la capacidad de visualizar únicamente imágenes simples (del tipo GIF o JPEG), es necesario un procesamiento previo de los datos vectoriales y raster servidos por un WFS y WCS respectivamente, para poder ser visualizados correctamente en el cliente. Existen un par de especificaciones complementarias que se encargan justo de la conversión de datos vectoriales y raster a formatos GIF o JPEG: Styled Layer Descriptor (SLD) [107] para la conversión en un WFS y Coverage Portrayal Service (CPS) [106] para un WCS.

Para fomentar la integración e interoperabilidad con servicios web no geo-espaciales, OGC está adoptando las especificaciones WMS, WFS, y WCS al contexto de los servicios web. El conjunto de experimentos de interoperabilidad denominados OGC Web Services (OWS), están encargado del proceso de migración de la información geo-espacial de los tres servicios básicos a una representación XML equivalente. También, OGC junto con el ISO Technical Committee 211 (ISO TC211), han elaboraron una norma estándar que describe una arquitectura basada en servicios web para la interoperabilidad de servicios

geo-espaciales OWS [70]. En otro esfuerzo para conformar sus especificaciones al mundo de los servicios web, OGC publicó recientemente el Catalogue Services [110] para la publicación y descubrimiento de datos geo-espaciales y servicios OWS. Esta especificación añade el rol de registro de servicios ausente en las primeras especificaciones (únicamente contemplaban interacción cliente-proveedor) y facilita el modelo de referencia funcional de los servicios web: publicar-buscar-enlazar.

El auge y expansión de los servicios web está produciendo que muchos modelos de sistemas información existentes migren o se adopten a este nuevo modelo, para aprovechar una de las grandes ventajas que los servicios web (y la arquitectura SOA en general) proporcionan: la composición de servicios web.

2.5. Composición de servicios web

La composición de servicios web es una de las facetas más importantes de la capa *Processes* de la pila de servicios web (véase Figura 2.2). Realmente, la idea detrás de la composición de servicios web es simple. En esencia, los usuarios interactúan con servicios web disponibles en Internet. Pero en la composición de servicios son los propios servicios web los que interactúan entre ellos para formar nuevos servicios. La finalidad de la composición de servicios web consiste pues en la creación de nuevos servicios a medida a partir de servicios web existentes. La siguiente pregunta es obvia: ¿cómo y qué mecanismos permiten la composición de servicios web?. Como hemos visto durante este capítulo, la idea de la composición no es un concepto nuevo. Los componentes software (véase sección 2.2) proponen el desarrollo de aplicaciones software mediante la composición de módulos de componentes software. Similarmente, los workflows (véase sección 2.3) son una colección (o composición) de actividades o tareas relacionadas entre sí mediante flujo de datos y de control. Ambas técnicas son *a priori* candidatas como mecanismos para la composición de servicios web. Pero también ambas técnicas fueron diseñadas teniendo en mente contextos Intranet, controlados y homogéneos, donde el proceso de composición es controlado completamente por un único participante. Sin embargo, la Web es un entorno completamente distinto ya que por naturaleza es dinámico y heterogéneo, donde no siempre es posible encontrar los servicios necesarios o éstos pueden aparecer y desaparecer sin previo aviso. Las técnicas tradicionales resultan insuficientes y poco flexibles para componer servicios web en el entorno de la Web, siendo entonces mucho más coherente adoptar tales técnicas a las características inherentes de la Web [135], cuestión que abordaremos en el capítulo 3.

Es necesario aclarar terminología antes de pasar a revisar el estado de arte en la composición de servicios web. Los términos orquestación (*orchestration*) y coreografía (*choreography*) describen dos aspectos distintos de la composición de servicios web [124]. La Figura 2.4, extraída de [124], ilustra las diferencias y relaciones entre ambos aspectos. La orquestación hace hincapié en la composi-

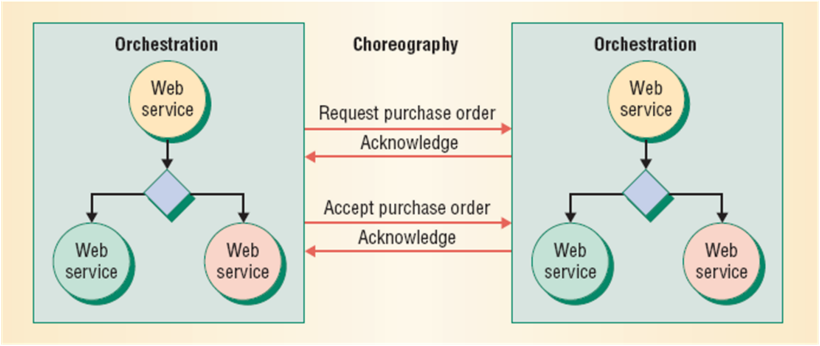


Figura 2.4: Coreografía vs. orquestación

ción de servicios web vistos como procesos ejecutables, privados y controlados por un único participante. La coreografía representa la colaboración entre tales procesos, centrándose en los mensajes públicos existentes entre los diferentes participantes de un proceso colaborativo. Ya que esta tesis se centra exclusivamente en la orquestación, el resto de esta sección revisa el estado de arte en el contexto de la orquestación de servicios web, tanto desde el punto de vista comercial (véase sección 2.5.1) como académico (véase sección 2.5.2).

2.5.1. OASIS WSBPEL

La especificación OASIS Web Services Business Process Execution Language (WSBPEL) [105] puede considerarse en estos momentos como el estándar *de facto* de la industria para la composición de servicios web basado en procesos (o workflows). WSBPEL contiene procesos abstractos y ejecutables. Los abstractos tratan con la coreografía, mientras que los ejecutables se centran en la orquestación de servicios web, el foco de interés de esta tesis. Conceptualmente, un proceso ejecutable está compuesto de actividades, que pueden ser a su vez básicas o estructuradas. Las actividades básicas representan invocaciones síncronas o asíncronas a servicios web o transferencias de datos entre las variables globales definidas en el proceso. Por ejemplo, las actividades básicas `invoke`, `reply` y `receive` interactúan con servicios web mientras que actividades `assign` se encargan del flujo de datos entre servicios web. Además, estas actividades básicas pueden estructurarse y componerse para formar procesos mediante el uso de las actividades estructuradas que definen el flujo de control. Algunos ejemplos de actividades estructuradas son: `sequence` que define una secuencia ordenadas de actividades; `flow` para ejecutar varias actividades en paralelo; `if`, describe un flujo de control condicional; o `forEach` para iterar ciertas veces sobre una o varias actividades. WSBPEL también proporciona mecanismos para la gestión de errores al estilo del lenguaje de programa JAVA (con construcciones como `throw`, `catch`, y `faultHandler`).

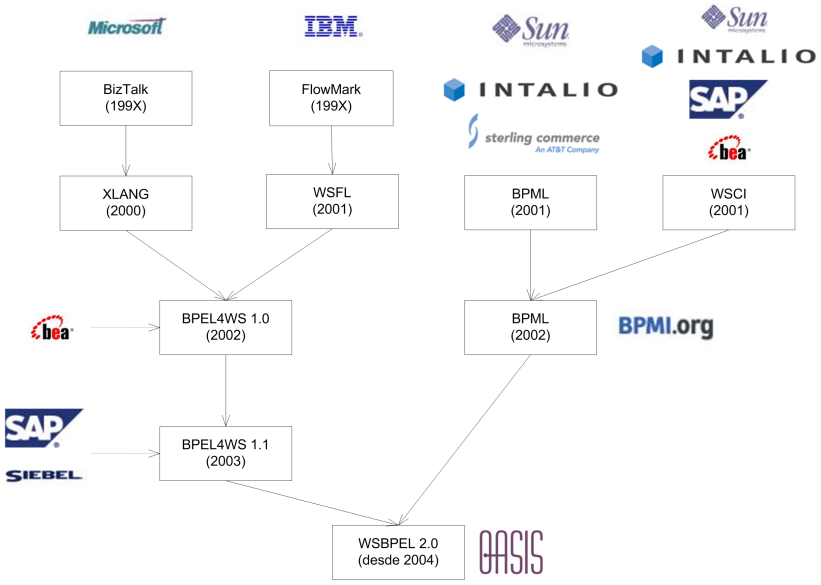


Figura 2.5: Evolución del lenguaje OASIS WSBPEL

La Figura 2.5 intenta mostrar la evolución de los lenguajes de composición de servicios web hasta la versión actual de WSBPEL. La última versión disponible (2.0) se encuentra bajo el auspicio de OASIS, aunque durante la redacción de esta tesis todavía se trata de un documento de trabajo por lo que aún puede estar sujeto a cambios y modificaciones antes de su publicación definitiva. WSBPEL, un lenguaje basado en XML (un ejemplo de código WSBPEL puede encontrarse en el Anexo B), fue originariamente creado como fusión de dos lenguajes anteriores y bastante distintos entre sí: WSFL y XLANG. El lenguaje WSFL utiliza unos constructores basados en grafos para el flujo de control mientras que XLANG sigue un estilo basado en bloques o estructural. [5, 124] proporcionan más información sobre estas dos especificaciones ya obsoletas. WSBPEL adoptó ambos estilos para definir el flujo de control de un proceso en vez seleccionar las mejores características de cada uno. Esto ha hecho de WSBPEL un lenguaje bastante complejo por la cantidad de constructores alternativos y redundantes que posee [121].

WSBPEL ha servido de base para experimentar la integración tanto de características de la web semántica [89, 125] como de reglas de negocio embebidas en el propio lenguaje [131]. También, ha sido objeto de estudio como modelo de composición [75] y puesto en práctica para la generación de servicios web médicos [9, 10]. WSBPEL puede considerarse como la aportación de la industria a la composición de servicios web. Sin embargo, existen multitud de otras técnicas para la composición de servicios web como muestra la sección siguiente.

2.5.2. Trabajos relacionados

A pesar del indudable avance que WSBPEL ha supuesto para la composición de servicios web, todavía existen problemas y aspectos por resolver en el ámbito de la composición, como aspectos semánticos, de calidad de servicio (QoS) o flexibilidad de la composición entre otros [139, 157]. Esta sección intenta mostrar una representación variada de trabajos sobre la composición de servicios web desde perspectivas diferentes, resumidos en la Tabla 2.2, junto con las semejanzas y diferencias con nuestro trabajo.

Tipo de trabajo	**Referencia**
Aproximaciones generales	[14, 18, 19, 20, 34, 47, 151]
Patrones workflow	[17, 16, 71, 96, 147]
Componentes software	[162, 163, 164]
Model Driven Architecture (MDA)	[10, 12, 62]
Quality of Service (QoS)	[32, 98, 114, 165]
En el contexto geográfico	[6, 8, 22, 64, 77, 84, 87, 114]

Tabla 2.2: Literatura sobre composición de servicios web

Aproximaciones generales

Calificamos los siguientes trabajos como *aproximaciones generales* porque abarcan no solo la composición sino también otros aspectos de los servicios web como la búsqueda, gestión, registro o ejecución.

En [14] se describe un sistema de composición de servicios web basado en el concepto de *service domain* y el modelo de mediadores. Un *service domain* representa una comunidad de servicios web relacionados entre sí que son accedidos mediante un servicio web especial que actúa de mediador, denominado *mediator service*. Este servicio mediador es el encargado de proporcionar los servicios web concretos dada una petición, mediante ciertos algoritmos de selección de servicios basados en un lenguaje de reglas deductivas. El concepto de *mediator service* y de componente integrado tienen similitudes en el sentido que representan a un conjunto de servicios web relacionados, pero en nuestro trabajo, un componente integrado no realiza las funciones típicas de mediador.

Self-Serv [18, 20] propone un lenguaje para componer servicios web basados en diagramas de estado. Un diagrama de estado está compuesto de estados y transiciones. Un estado puede ser simple o compuesto, donde los simples son invocaciones a servicios (simple o composiciones) y los compuestos estructuran diagramas de estados anidados. Las relaciones entre estados se interpretan como transiciones que son activadas por eventos. *Self-Serv* define un modelo de ejecución *peer-to-peer* donde la coordinación de la ejecución de la composición

(diagrama de estado) se distribuye entre los servicios remotos (estados). Posteriormente, *Self-Serv* ha sido ampliado en [19] con una herramienta middleware para la generación de composiciones de servicios web a partir de esquemas conceptuales como los diagramas de estado. La mayor parte de este trabajo está dedicada a la ejecución distribuida de composiciones, a diferencia de nuestro trabajo que parte de la premisa de la ejecución centralizada.

eFlow [34] es una plataforma que soporta la composición dinámica y gestión de servicios compuestos. Un servicio compuesto se interpreta como un grafo que define el orden de ejecución de los nodos de la composición. Estos nodos pueden ser de tipo servicio, decisión o evento. Los nodos servicio representan invocaciones de servicios básicos o compuestos. Este tipo de nodos contiene una consulta que selecciona en tiempo de ejecución el servicio web que será invocado. Los nodos decisión representan el flujo de control mientras que los nodos evento actúan como un sistema de notificación, permitiendo el envío y recepción de diferentes tipos de eventos. La característica más interesante de *eFlow* es su flexibilidad ante cambios del entorno debido a que los nodos servicio no están ligados a instancias de servicios web concretas en tiempo de diseño. Esta característica también está presente en nuestro trabajo, ya que la invocación de un componente integrado no implica la invocación de un servicio web concreto. Pero además, simplificamos el modelo de componente ya que únicamente existe un tipo de componente en nuestro modelo en vez de tres tipos como en *eFlow*.

WSMF [47] define un modelo conceptual para la descripción y composición de servicios web semánticos. El modelo WSMF se basa en el principio de maximizar tanto el desacoplamiento entre los elementos básicos que forman el modelo como el uso de mediadores [158] para conseguir la máxima escalabilidad entre dichos elementos. Concretamente, WSMF está formado por cuatro elementos básicos: ontologías que proporcionan la terminología utilizada por los restantes elementos; objetivos que definen los problemas que van a resolver los servicios web; las descripciones de servicios web que especifican las capacidades de los propios servicios web; y los mediadores que permiten que los diferentes elementos pueden interactuar entre ellos, garantizando la interoperabilidad entre estos elementos. Sin embargo esta iniciativa fue desarrollada únicamente a nivel conceptual, sirviendo de punto de partida para definir la ontología WSMO [45, 44] definida para describir diversos aspectos de los servicios web semánticos, refinando el marco conceptual de WSMF mediante el desarrollo de una ontología y de un lenguaje formal. El modelo conceptual WSMF [47] y nuestro trabajo están situados en contextos distintos, el primero orientado hacia la composición de servicios web semánticos mientra que nuestro trabajo persigue la reutilización de composiciones de servicios web.

METEOR-S [151] es un proyecto que intenta aplicar las características de la web semántica y los servicios web en la gestión de procesos. Son tratados diversos aspectos como el desarrollo, composición, publicación y descubrimiento de servicios web semánticos. La contribución más destacable de METEOR-S ha sido proporcionar una infraestructura *peer-to-peer* para la publicación y descubri-

miento de servicios web semánticos basado en el "enriquecimiento semántico" de los estándares para los servicios web como WSDL y UDDI. Como resultado, METEOR-S ha producido el lenguaje WSDL-S [138], un mecanismo para la anotación semántica de descripciones de servicios web más sencillo que las ontologías para la descripción de servicios web semánticos (como OWL-S o WSMO). Nuestro trabajo adoptará WSDL-S para la descripción de componentes integrados utilizando conceptos definidos en ontologías de dominio.

Composición basada en patrones de workflow

Benatallah et al. [17, 16] proponen un catálogo de patrones para describir la composición, descubrimiento y ejecución de servicios web. Se trata de patrones que describen interacciones entre servicios web a alto nivel de abstracción. Por ejemplo, el patrón de composición describe relaciones de larga duración entre los participantes que intervienen en un proceso de composición o el patrón de descubrimiento trata con la integración dinámica de servicios web como consecuencia de un proceso automático de búsqueda e integración. Ninguno de estos patrones abordan directamente el flujo de control entre servicios web o enfatizan en la reutilización de éstos, mas bien se centran en aspectos como la colaboración y coreografía entre los participantes de un proceso.

La aproximación basada en patrones de Jaeger et al. [71] propone un modelo abstracto usando patrones de workflow para la agregación de propiedades de calidad de servicio. Como en nuestro trabajo, los patrones de workflow propuestos son derivados de los patrones tradicionales de van der Aalst et al. [148]. Sin embargo, a diferencia de nuestro trabajo, la interpretación de los patrones es completamente diferente ya que los autores [71] utilizan los patrones para la agregación de dimensiones de calidad de servicio mientras que nuestros patrones enfatizan la reutilización de servicios web.

El trabajo de Melloul et al. [96] propone algunas ideas interesantes en su aproximación para la composición de servicios web basada en identificar patrones expresados a alto nivel como objetos que pueden ser especializados y reutilizados para la construcción de otros patrones. Su concepto de patrón es similar a una función que embebe llamadas a otras funciones. Aunque en nuestro trabajo los usuarios necesitan conocer los servicios web que van a ser utilizados en la composición, difiere en que nuestros patrones permiten expresar mayor variedad de constructores para el flujo de control que simplemente la especialización de patrones sin estructuras de flujo de control explícitas.

El objetivo del trabajo de Tut et al. [147] se centra en el estudio de patrones genéricos para la composición de servicios web. Sin embargo, el concepto de patrón propuesto puede interpretarse como una plantilla para procesos genéricos que satisface requisitos de usuario específicos, en vez de patrones para la especificación de pasos concretos para llevar a cabo la composición y reutilización de servicios web en un contexto determinado, tal como propone nuestro trabajo.

Composición basada en componentes software

Escasos trabajos tratan la composición de servicios web desde la perspectiva de los componentes software, a pesar de que algunos reconocidos expertos en el campo de los servicios web identifican los servicios web como un tipo particular de componentes reutilizables [40]. Los trabajos de Yang et al. [162, 163, 164] pueden considerarse pioneros en la componentización de servicios web. Los autores tratan los servicios web como componentes software con la idea de soportar los principios básicos del desarrollo de software como la reutilización, especialización y extensión. La idea principal consiste en la encapsulación de la lógica de la composición dentro de una definición de clase, que representa un componente web. Estos componentes web pueden ser especializados o extendidos para crear nuevos componentes web. Este concepto tiene similitudes evidentes con nuestro concepto de componente integrado ya que ambos trabajos están influidos por la idea común de los componentes software. Sin embargo, nuestro trabajo complementa la idea de componente integrado aportando mayor variedad de patrones de workflow para la definición del flujo de control en composiciones de servicios web.

Composición basada en MDA

Anzböck et al. [10] proponen una aproximación MDA para la composición de servicios web en el contexto médico produciendo como resultado procesos WSBPEL. La finalidad de los autores es reducir la complejidad inherente de los workflows en el campo de la medicina, para permitir mediante transformaciones manuales y automáticas la creación de composiciones de servicios web, añadiendo también requisitos transaccionales y de seguridad necesarios en este determinado contexto. Este trabajo es uno de las primeras aplicaciones de la composición de servicios web al campo de la medicina y, en particular, tiene que tratar con los protocolos y estándares concretos de ese contexto. El aspecto MDA de este trabajo radica en que facilita la conversión de los protocolos típicos para workflow en el campo de la medicina a los servicios web.

El proceso de desarrollo de plantillas (*skeletons*) de composiciones de servicios web siguiendo una aproximación MDA centra la propuesta de Baïna et al. [12]. Los autores toman como punto de partida las especificaciones externas de los servicios web (por ejemplo interfaces y protocolos) para generar las plantillas de composiciones de servicios web. Posteriormente, los diseñadores puede completar tales plantillas con la lógica de la aplicación necesaria para crear finalmente un proceso ejecutable concreto. Este proceso es similar, por ejemplo, a la generación de código siguiendo una aproximación MDA donde el programador debe completar el código inicial del conjunto de clases e interfaces generadas.

Grønmo et al. [62] introducen una metodología que utiliza UML (Unified Modeling Language) y MDA para la composición de servicios web. Por una parte, UML proporciona un alto nivel de abstracción de la lógica de la composición

de servicios para la generación de especificaciones ejecutables en diferentes lenguajes de composición como WSBPEL. De esta forma, el modelo conceptual del proceso es independiente de la representación concreta utilizada para el proceso ejecutable. Concretamente, los autores proponen el desarrollo de composiciones de servicios web combinando UML y WSDL. Las descripciones WSDL de los servicios web son convertidas a UML. Luego, los modelos UML son integrados para formar servicios web compuestos. Finalmente, las descripciones UML de los nuevos servicios compuestos son exportados a un lenguaje de composición de servicios web.

Nuestro trabajo es similar a los anteriores trabajos que aplican el paradigma MDA como mecanismo para la composición de servicios web en el sentido que a partir de un diseño a alto nivel de abstracción (como los componentes integrados) se deriva de forma semi-automática el correspondiente proceso ejecutable (en formato WSBPEL).

Composición basada en QoS

Los siguientes trabajos abordan la calidad de servicio en las composiciones de servicios web. Los atributos de calidad asociados a los servicios web sirven para discernir, de entre un conjunto de servicios web idénticos funcionalmente, el servicio que mejor se adapta a los requisitos de un usuario. Aunque no tratan directamente la problemática de la reutilización de servicios web, son interesantes mencionarlos como aproximaciones para la composición de servicios web desde una perspectiva diferente.

Cardoso et al. [32] aborda el problema de establecer métricas de calidad adecuadas para servicios entre los participantes de un proceso colaborativo. Los autores presentan un modelo de calidad de servicio que permite establecer la calidad de servicio de workflows de forma automática a partir de los atributos de calidad de las tareas individuales. Similarmente, Menascé [98] expone ciertos atributos cualitativos y cuantitativos de calidad de servicio que deberían estar presentes en la composición de servicios web. En particular, en la composición de servicios web se requiere que todas las partes implicadas (clientes y proveedores) lleguen a un acuerdo acerca de los requisitos funcionales y no funcionales, incluyendo coste, disponibilidad o rendimientos de los servicios web. Onchanga [114] también propone un modelo de calidad de servicio pero orientado al contexto de la composición de servicios web geográficos, aportando ciertos atributos de calidad críticos para la información geográfica, como la actualización de los datos geográficos (*freshness*). Finalmente, el trabajo propuesto por Zeng et al. [165] presenta una plataforma middleware para la adecuada selección de servicios web que formarán parte de una composición de servicios en función de que éstos maximizan una función de utilidad expresada en atributos de calidad de servicio.

En general, los atributos de calidad serán necesarios para ajustar con mayor detalle los requisitos del usuario a las capacidades de los servicios web.

En nuestro trabajo no añadimos explícitamente atributos de calidad de servicio, aunque mediante los patrones de selección propuestos permitimos que el servicio web finalmente seleccionado sea el que se ejecuta con mayor rapidez. Implícitamente, nuestro trabajo considera el tiempo de respuesta como atributo de calidad.

Trabajos de composición en el contexto geográfico

La interoperabilidad entre sistemas de información geográfica es un requisito necesario para permitir el acceso e integración de fuentes de datos heterogéneas [2, 25, 133]. En esta línea, la composición de servicios web se ha convertido en un paradigma esencial que proporciona flexibilidad a los sistemas de información geográfica y facilita el acceso a información geográfica. A continuación, describimos algunos trabajos interesantes para la composición de servicios geo-espaciales ya que nuestro trabajo ha sido desarrollado en parte en el contexto de los servicios web geo-espaciales.

El trabajo de Alameh [6] fue uno de los primeros intentos en describir la composición de servicios web geográficos. Básicamente, este trabajo explora conceptualmente diferentes arquitecturas para la composición de servicios web geográficos atendiendo a la clasificación propuesta por ISO 19119 [70]. Sin embargo, los lenguajes de composición actuales eran relativamente inmaduros o existían demasiadas alternativas en competencia sin un claro ganador (véase Figura 2.5).

An et al. [8] proponen un patrón para geo-servicios basado en ontologías para la publicación de geo-servicios que posteriormente serán descubiertos utilizando mecanismos basados en semántica. Los objetivos son distintos porque este trabajo [8] se centra principalmente en procesos de descubrimiento de geo-servicios mientras que la composición y reutilización son el foco de nuestro trabajo.

Bernard et al. [22] describen el problema de la interoperabilidad en la composición de servicios web geográficos y plantean cuestiones abiertas que aún deben resolverse. Una de las limitaciones que encuentran los autores es la falta de flexibilidad en la composición de servicios geo-espaciales, precisamente uno de los objetivos abordados en nuestro trabajo.

Bastante interesante es el framework para servicios geo-espaciales presentado por Kim et al. [77, 84], el cual se centra en la eficacia para manejar grandes volúmenes de documentos GML [109] mediante servicios web. En realidad, una de las características que diferencia los servicios geo-espaciales frente a servicios web en otros contextos radica en la cantidad de información, bien en formato GML o como imágenes binarias, que llegan a intercambiar o manejar los servicios geo-espaciales. Aunque resulta un reto interesante analizar el rendimiento en la trasmisión de diferentes formatos, visto como un atributo de calidad de servicio, nuestro trabajo ha versado hacia objetivos completamente diferentes.

Finalmente, Li et al. [87] presentan los resultados de la investigación acerca

del modelado distribuido de workflows para datos geográficos. Los autores examinan otros aspectos menos tecnológicos relacionados con los workflows como por ejemplo las estructuras organizacionales necesarias o los diferentes tipos de trabajadores involucrados para poder poner en marcha ese tipo de workflows.

En conclusión, tanto los workflows como la composición de servicios web está siendo investigada en el campo de los sistemas de información geográfica desde diferentes ángulos. Sin embargo, ninguna de las aproximaciones descritas anteriormente ha tratado de forma directa la problemática de la reutilización como mecanismo para componer servicios web más eficientemente.

2.6. Reutilización de servicios web

La sección anterior ha proporcionado una amplía visión de la problemática que envuelve la composición de servicios web. En esta sección pretendemos definir mejor los límites del problema que queremos resolver con nuestro trabajo centrándonos en la reutilización de servicios web. La reutilización es un importante aspecto dentro de la composición porque intenta mejorar el proceso de composición convirtiéndolo en un proceso más eficiente y simple. Sin embargo aún existen pocos trabajos que tratan directamente con la reutilización de servicios web, como se describe a continuación.

2.6.1. Trabajos relacionados

Henderson et al. [67] proponen un arquitectura para la reutilización basada en servicios web sin estado y en mensajes con estado, donde las interacciones asíncronas y de larga duración son comunes. El reto propuesto es reemplazar, como si de componentes software se trataran, servicios web durante interacciones asíncronas en marcha. Sin embargo, este tipo de escenarios requiere que los servicios web tengan asociado un estado que indique el progreso de la interacción. Partiendo de la premisa de que los servicios web no tienen estado, es posible asociar cierto estado a un servicio web bien utilizando un recurso externo e independiente o bien con un mensaje embebido en la propia interacción asíncrona. Los autores proponen el segundo mecanismo, mensajes con estado, para describir el comportamiento de los servicios web, aspecto necesario para convertir los servicios web en escenarios asíncronos en componentes reutilizables. Nuestro trabajo difiere en dos aspectos clave respecto al propuesto por [67]. En primer lugar, nuestro escenario envuelve principalmente interacciones síncronas entre servicios web. En segundo lugar, nuestro enfoque de reutilización se entiende como mecanismo de composición mientras que en [67] se orienta hacia la capacidad de reemplazar un componente por otro.

Medeiros et al. [93] proponen una interesante aproximación para la anotación y reutilización de workflows científicos. Los autores presentan WOODSS, una infraestructura para que los científicos puedan especificar y anotar sus experimentos y actividades científicas. El objetivo principal es la reutilización de

workflows y para ello los autores definen el concepto de *digital content components* (DCC), unidades reutilizables para encapsular workflows anotados. El DCC es similar a nuestro concepto de componente integrado ya que ambos son las unidades básicas reutilizables para formar composiciones de servicios web y finalmente convertirlas en un proceso WSBPEL ejecutable. Sin embargo, las unidades DCC son compuestas manteniendo los mismos constructores que proporciona WSBPEL. Como veremos a lo largo de esta tesis, nuestro conjunto de patrones evita redundancias y solapamientos como ocurre en el caso de WSBPEL (véase sección 2.5.1). En este sentido, nuestro trabajo proporcionará un mecanismo de composición y reutilización más simple y estructurado.

La visión de reutilización de Pautasso et al. [122] se centra en la posibilidad de utilizar diferentes servicios web para un mismo componente. Los autores fijan su atención en la declaración de enlaces flexibles que únicamente definen la mínima sintaxis y semántica de los servicios web que van a ser compuestos. La idea clave reside en retrasar lo máximo posible, idealmente hasta el tiempo de ejecución, la elección del servicio web que finalmente será ejecutado. Nuestro trabajo también tiene en cuenta este aspecto de reutilización, que denominaremos flexibilidad, permitiendo que la elección del servicio web que ejecutará realmente la funcionalidad de un componente integrado sea retrasada hasta el tiempo de ejecución de la composición.

El trabajo de Voisard et al. [156] propone una arquitectura en capas para el diseño de sistemas de información geográfica interoperables. Cada capa representa un nivel diferente de abstracción, empezando desde la capa de aplicación hasta llegar a la capa de invocación de los servicios. Las ideas expuestas en [156] han sido adoptadas en el desarrollo de nuestro trabajo ya que los componentes integrados formarán el nivel de servicios abstractos en una arquitectura dividida por niveles de abstracción (véase capítulo 4).

Los siguiente trabajos [96, 163, 164] ya han sido descritos anteriormente en la sección 2.5.2, ya que se basan principalmente en la composición de servicios web aunque tratan también ciertos aspectos de la reutilización de servicios web.

2.7. Resumen

El éxito inicial de la Web puede atribuirse sin lugar a duda a su simplicidad [43], proporcionando un mecanismo simple para que la información estática sea publicada y compartida de forma global. Los servicios web añaden una capa de funcionalidad encima de la Web estática que puede ser utilizada por cualquiera. En esencia, los servicios web han sido diseñados para ser reutilizados.

Sin embargo, aunque existen gran variedad de trabajos y lenguajes para la composición de servicios web, pocos tratan de manera directa la problemática de la reutilización para proporcionar mecanismos más eficientes y simples para crear composiciones. Además, como hemos visto en la sección 2.5 y 2.6, casi todos los trabajos explotan bien el concepto de componentes software o bien

los patrones de workflow, pero no los dos a la vez. A lo largo de esta tesis trataremos de describir nuestro modelo basado en componentes integrados para la reutilización de servicios web, que integrará características de los componentes software, de los patrones de workflow y de los servicios web.

Capítulo 3

Componentes integrados

En este capítulo describimos el concepto de componente integrado, aspecto clave que proporciona las bases para la definición de un modelo de composición de servicios web que facilita la reutilización como mecanismo de composición. Un componente integrado combina características de los componentes software, de los patrones de workflow y de los servicios web, separando en dos planos distintos la funcionalidad que ofrece un componente integrado de cómo la ofrece. A partir del concepto de componente integrado, en el capítulo siguiente mostraremos una metodología para la creación, composición y reutilización de dichos componentes integrados.

3.1. Introducción

La reutilización de software tiene una larga historia en el área de la ingeniería del software [24] y en los sistemas basados en componentes [142], aunque todavía puede considerarse una área activa de investigación en nuestro días [39]. Entre las numerosas definiciones del concepto de reutilización disponibles en la literatura, hemos seleccionado la proporcionada por Wiederhold et al. [160] ya que se trata de una definición que transmite claramente la noción de reutilización buscada en esta tesis:

> *Use and reuse become indistinguishable.*

Básicamente, éste es el objetivo fundamental de un componente integrado. Definiremos un componente integrado como el bloque fundamental o unidad básica reutilizable de tal forma que *utilizar* o *reutilizar* un componente integrado

sea indistinguible, proporcionando además mayor flexibilidad, en el sentido de ofrecer mayores opciones de selección. El hecho de aumentar el nivel de reutilización mediante componentes integrados facilitará notablemente el desarrollo de aplicaciones web [7].

El principio básico de los sistemas basados en componentes se centra en la construcción de aplicaciones mediante la composición de componentes software reutilizables [24, 142]. Atendiendo a esta definición, el estudio de la reutilización de servicios web llevado a cabo en esta tesis parte obligatoriamente de la definición de componentes software y de los propios sistemas basados en componentes. Existe una extensa literatura sobre este tipo de sistemas pero básicamente pueden ser caracterizados según su modelo de componente y la metodología de composición [24, 142].

- El *modelo de componente* define cómo describir los componentes que forman el sistema con el fin de mejorar su reutilización y el nivel de sustitución, es decir, la capacidad de reemplazar un componente por otro funcionalmente similar sin producir cambios en el sistema. Algunas características que afectan al nivel de reutilización en el modelo de componente pueden ser por ejemplo el nivel de granularidad del componente, la encapsulación o la estandarización de las interfaces.

- La *metodología de composición* describe los mecanismos y técnicas necesarias para combinar los componentes definidos en el modelo de componente. Estas técnicas cubren aspectos puramente de composición como el flujo de datos entre diferentes componentes o el orden en que se combinan tales componentes.

En esta tesis partimos de estos dos principios básicos de los sistemas basados en componentes para establecer las bases de la reutilización de servicios web. El concepto de componente integrado, desarrollado a lo largo de este capítulo, se corresponde con el modelo de componente de los sistemas basados en componentes. Un componente integrado reúne características de los servicios web [7], de los sistemas basados en componentes [143], y de los patrones de workflow [148]. La combinación apropiada de las características de estas áreas dará lugar al concepto de componente integrado. De forma similar proponemos una metodología de composición como mecanismo para gestionar tales componentes integrados y definir cómo pueden combinarse. El capítulo 4 se dedicará íntegramente a la descripción de esta metodología de composición.

La Figura 3.1 ilustra gráficamente la influencia que ejercen los servicios web, los sistemas basados en componentes y los patrones de workflow en la definición de un componente integrado. En general, la idea del modelo de componente se corresponde con un componente integrado visto como una unidad básica reutilizable. Ya que esta tesis se desarrolla en el contexto de servicios web, nos interesa que un componente integrado tenga la misma apariencia externa que un servicio web. Para ello, adoptaremos los estándares para describir interfaces

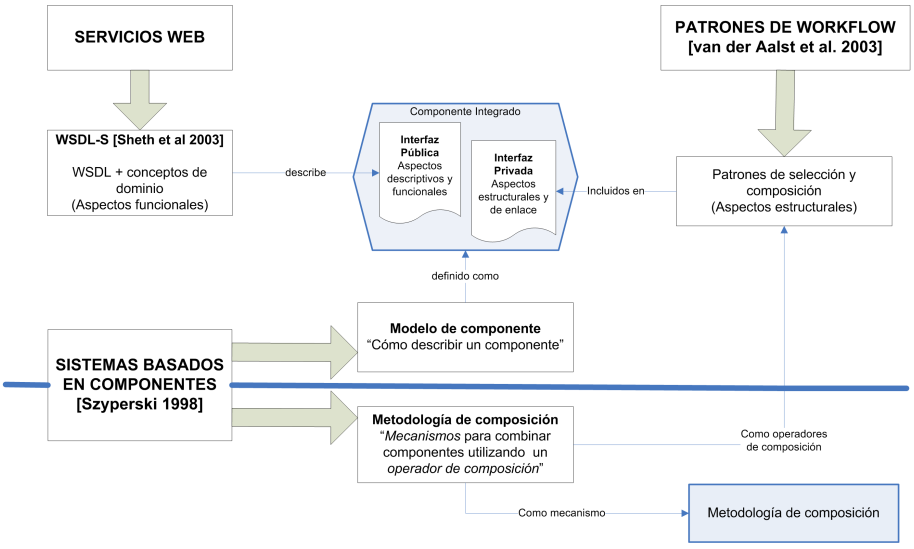


Figura 3.1: Áreas influyentes en un componente integrado

de servicios web. Por otra parte, los patrones de workflow nos proporcionarán mecanismos para combinar componentes integrados ya existentes con el fin de crear otros nuevos. Este capítulo se centra en todos los aspectos que intervienen en la definición de un componente integrado, justo la parte de arriba de la línea horizontal que aparece en la Figura 3.1. La parte de abajo, la metodología de composición, será tratada en el siguiente capítulo.

3.2. Definiendo un componente integrado

Esta sección presenta el concepto de componente integrado, tanto desde el punto de vista del contexto en el que tiene lugar como desde la descripción de su organización interna.

3.2.1. Componente software vs. servicio web

El primer punto a tener en cuenta es porqué hemos denominado a un componente integrado de este modo. ¿Se trata de un componente software?. O por el contrario, ¿es un concepto más cercano a la definición de un servicio web?. Ha habido bastante debate acerca de las diferencias entre componente software, y por extensión la arquitectura basada en componentes, y los servicios web, y la arquitectura SOA.

La arquitectura basada en componentes y SOA son diferentes ya que tratan de resolver diferentes problemas desde diferentes puntos de vista. SOA es una

arquitectura orientada a servicios, donde la composición de servicios es una tarea completamente distinta a enlazar algunos componentes juntos en una aplicación en el contexto de una Intranet. En el mundo de los servicios web no existe compilación, no existe un contexto particular de ejecución ni enlaces de código como ocurre con los componentes software. La utilización de servicios web se basa simplemente en invocarlos directamente, ya que no mantienen un estado interno del contexto particular en el que son ejecutados. Por ejemplo, un servicio web no tiene variables internas con distintos valores para cada instancia como ocurre con los componentes software. Por este motivo, reemplazar un componente con estado por otro es siempre mucho más complejo que reemplazar pares de componentes sin estado. La ausencia de estado es una de las razones por la que los servicios web no deben confundirse con componentes u objetos distribuidos [155].

El diseño y la construcción de servicios en SOA es diferente a la de los componentes. Sin embargo, desde la perspectiva del proveedor de servicios, la línea que separa un servicio de un componente software no se muestra siempre tan clara. Es perfectamente posible disponer de servicios que de hecho son implementados como componentes software. Desde el punto de vista SOA, la implementación concreta de un servicio es irrelevante (componentes, objetos, bases de datos o sistemas *legacy*), lo verdaderamente importante son las descripciones de las interfaces de los servicios.

Respecto a la expresión componente integrado, el término *componente* intenta enfatizar la característica de reutilización, inherente en los componentes software, en el contexto de los servicios web. La reutilización de código es una característica común en los desarrollos de software basados en componentes. Sin embargo, la reutilización no es tan habitual en aplicaciones en el contexto SOA. Evidentemente, no tratamos de reutilizar código sino descripciones de servicios, ya que únicamente tratamos a nivel de las descripciones de servicios.

El calificativo *integrado* está prestado de la iniciativa europea para el desarrollo de una Infraestructura de Datos Espaciales en Europa (INSPIRE[1]). El objetivo que persigue INSPIRE es crear una directiva marco que guíe e instruye a los estados miembros sobre la creación de infraestructuras de información espacial a nivel nacional y local que proporcione a cualquier tipo de usuarios servicios integrados de información geo-espacial. De esta definición destacamos los términos *servicios integrados*. Los usuarios buscan servicios que se adapten a sus requisitos. Normalmente no existe un único servicio que satisface completamente sus demandas, sino varios servicios individuales que encadenados forman una composición. El usuario recibe dicha composición de servicios pero con la apariencia de un único servicio integrado de información geo-espacial (un servicio opaco según la norma ISO 19119 [70]). Como veremos a lo largo de esta sección, un componente integrado esconde toda su complejidad como si se tratase de un único servicio web, tal como ocurre con el servicio integrado de información geo-espacial propuesto por la recomendación INSPIRE.

[1] `http://inspire.jrc.it`

3.2.2. Aspectos de un componente integrado

La definición de un componente integrado debe ser consistente con su fin, es decir, maximizar su reutilización. Es necesario considerar ciertas características deseables en un componente integrado, como que sea una unidad descriptiva de servicios compuestos e independiente de la composición de la cual forma parte, para poder ser reutilizada en otras composiciones. Además, también es una característica deseable en un modelo de componente que exista un único tipo de componente, es decir, que como resultado de combinar componentes integrados se obtenga otro componente integrado. El usuario final no debe conocer la complejidad interna de un componente integrado, ignorando si éste está compuesto a su vez de dos, cinco o diez servicios web.

En nuestro trabajo hemos diseñado un componente integrado como un conjunto de aspectos clave que nos permite perfilar su estructura, similar al conjunto de capas del modelo de referencia de los servicios web (véase capítulo 2), donde cada capa cubre diferentes aspectos de los servicios web. Además, queremos emular el comportamiento de un componente software para que sea independiente, reutilizable, con interfaces bien definidas y ofreciendo cierta encapsulación sobre los detalles de su estructura. Nuestra estrategia ha consistido en capturar de las capas *Description* y *Processes* de la pila de servicios web (véase Figura 2.2 del capítulo 2), todos los aspectos relevantes que definen un componente integrado para permitir su descripción, composición e invocación, con el fin de maximizar su reutilización. Los aspectos que definen a un componente integrado quedan agrupados en las siguientes cuatro categorías: aspectos *descriptivos*, *funcionales*, *estructurales*, y de *enlace*.

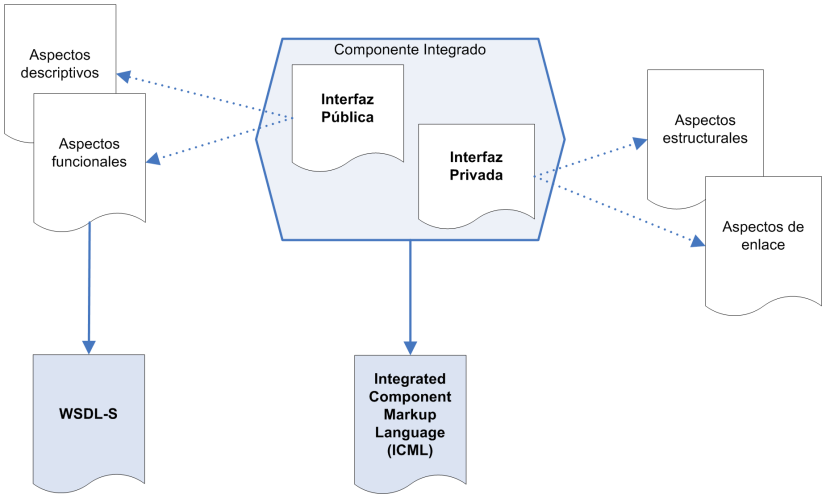


Figura 3.2: Aspectos de un componente integrado

La Figura 3.2 muestra la organización interna de un componente integrado. El hexágono[2] en la parte central de la figura representa un componente integrado. A ambos lados del componente integrado aparecen los diferentes aspectos que forman un componente integrado. De hecho, estos aspectos representan diferentes pero también complementarias vistas de un mismo componente integrado. Por ejemplo, los aspectos descriptivos y funcionales son útiles para la localización de componentes integrados porque expresan la funcionalidad que ofrece dicho componente integrado, información básica durante el proceso de búsqueda. Sin embargo, los aspectos estructurales y de enlace no aportan información relevante durante el proceso de búsqueda, mas bien, deberían ser desconocidos para un usuario que se encuentra buscando componentes integrados que se ajusten a sus necesidades. Para ofrecer este grado de encapsulación, hemos regulado el acceso a un componente integrado mediante interfaces funcionales (o vistas). Se trata de divisiones lógicas que agrupan los diferentes aspectos en una interfaz pública y otra privada, de forma similar a la encapsulación que proporcionan los lenguajes de programación orientados a objetos. La interfaz pública expresa públicamente los aspectos descriptivos y funcionales de un componente integrado, mientras que la interfaz privada representa una vista interna del componente integrado encapsulando los aspectos estructurales y de enlace. De esta forma, un usuario conoce únicamente la interfaz pública de un componente integrado, sin importarle los detalles de su estructura interna definidos en la interfaz privada.

Desde un punto de vista práctico, un componente integrado tiene asociado dos descripciones (ficheros). Todos los aspectos de un componente integrado se definen mediante una descripción basada en XML, Integrated Component Markup Language (ICML), desarrollada en esta tesis y cuyo esquema formal aparece en el Anexo A. Esta descripción no es visible al usuario final. Por otra parte, la funcionalidad de un componente integrado (aspectos funcionales) se describe en WSDL-S [138], un fichero WSDL anotado semánticamente, permitiendo explorar la interfaz funcional de un componente integrado como si se tratase de un servicio web. A continuación describimos los detalles de cada uno de los aspectos contenidos en un componente integrado.

Aspectos descriptivos

La finalidad de los aspectos descriptivos es actuar de metadatos referidos al contexto en el que tiene lugar un componente integrado. La Tabla 3.1 lista los atributos descriptivos. Por ejemplo, el atributo `ID` identifica unívocamente a un componente integrado. Generalmente, el valor de este atributo coincidirá con el nombre del fichero ICML que describe el componente integrado. El atributo `AccessPoint` almacena la URL que apunta a la descripción WSDL-S del componente integrado. Se trata pues del atributo que relaciona ambas descrip-

[2] Esta figura será utilizada a lo largo de esta tesis para representar a un componente integrado

ciones. Aunque la versión actual tan solo cuenta con el conjunto de atributos básicos de la Tabla 3.1, fácilmente pueden incorporarse otros atributos como el contexto comercial del componente integrado (mediante alguna clasificación comercial como United Nations Standard Products and Services Code[3]), el contexto geográfico (si cierto componente integrado sirve únicamente mapas de España, no tiene sentido por ejemplo utilizarlo en un escenario localizado en Noruega) o propiedades de pago electrónico. La lista de metadatos puede completarse con propiedades de calidad de servicio para refinar, por ejemplo, el proceso de búsqueda.

Atributo	Descripción
ID	Identificador único del componente integrado.
AccessPoint	URL del documento WSDL-S asociado.
Description	Descripción de la funcionalidad del componente integrado.

Tabla 3.1: Atributos descriptivos de un componente integrado

Aspectos funcionales

Los aspectos funcionales especifican la funcionalidad de un componente integrado detallando la operación que ofrece y sus parámetros de entrada y salida. Además de la funcionalidad sintáctica, los aspectos funcionales también incorporan ciertos aspectos semánticos. Tanto el nombre de la operación como cada uno de los parámetros de entrada y salida están anotados semánticamente mediante un concepto previamente definido en ontologías de dominio. La Figura 3.3 muestra como la relación entre las ontologías de dominio y un componente integrado se realiza a través de los aspectos funcionales. De esta forma, añadimos cierto grado de semántica a la descripción funcional de un componente integrado con dos objetivos en mente. Por una parte se describe con mayor precisión la funcionalidad que ofrece un componente integrado, refinando en consecuencia el proceso de búsqueda para el usuario. Por otra parte se logra cierto grado de automatización durante el proceso de composición ya que, por ejemplo, será posible establecer de forma automática flujos de datos entre parámetros atendiendo a su anotación semántica siempre que tales parámetros se refieran al mismo concepto. Evidentemente, no perseguimos en este trabajo un proceso automático de composición de servicios basado en la inferencia semántica. Simplemente, dejamos que se resuelvan ciertas conexiones siempre bajo la supervisión del diseñador (usuario). La sección 3.3 muestra un ejemplo de la descripción de los aspectos funcionales, integrando propiedades sintácticas y semánticas en una única descripción funcional.

[3] `http://www.unspsc.org/`

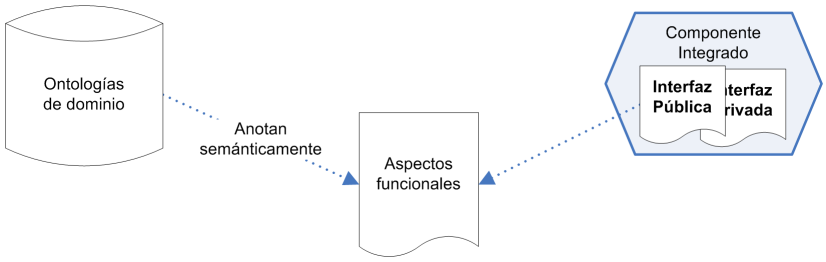


Figura 3.3: Aspectos funcionales de un componente integrado

Aspectos estructurales

Los aspectos estructurales especifican el flujo de control de un componente integrado definiendo su estructura interna como combinación de servicios web u otros componentes integrados más simples.

En general, el flujo de datos entre dos tareas pertenecientes a un proceso implica un determinado flujo de control, es decir, la especificación de un orden parcial de ejecución de tareas [65]. Por ejemplo, para que puedan transferirse datos de la tarea A a la B, es necesario que la tarea A finalice y que la tarea B no haya sido aún iniciada. De la especificación de los flujos de datos entre tareas se derivan patrones de workflow específicos que permiten al diseñador de procesos determinar el flujo de tareas de un proceso. La aportación del trabajo de van der Aalst et al. [148] ha consistido en proporcionar un conjunto coherente de patrones de workflow para modelar cualquier workflow por complejo que sea.

Nuestro trabajo ha consistido en la definición de un subconjunto de patrones [4] de los propuestos en [148] en el contexto de la composición de los servicios web. Concretamente, hemos analizado los patrones originales para encontrar un subconjunto que se adapte a las características de un componente integrado (independencia, encapsulación, etc.) y facilite la reutilización de éstos. La Figura 3.4 muestra como el patrón, descrito en los aspectos estructurales, organiza la estructura interna de un componente integrado. Además, dichos patrones no son visibles ya que los aspectos estructurales pertenecen a la interfaz privada de un componente integrado. El conjunto de patrones para los componentes integrados será analizado en la sección 3.5.

Aspectos de enlace

En escenarios reales, como por ejemplo en transferencias de conjuntos de datos GML, los servicios web suelen recibir grandes estructuras de datos en XML como entradas y producir posiblemente grandes documentos XML como salida. Los flujos de datos indican cómo se transfieren estos documentos XML entre diferentes servicios web, especificando las reglas de transferencia entre

[4] Utilizaremos a lo largo de esta tesis el término patrones en vez de patrones de workflow

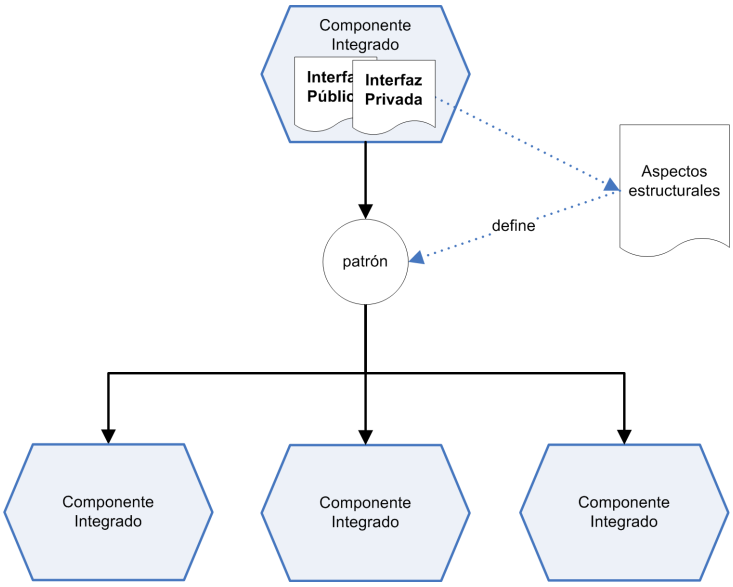


Figura 3.4: Aspectos estructurales de un componente integrado

los parámetros de salida y entrada. Los aspectos de enlace están íntimamente relacionados con los aspectos estructurales ya que de forma conjunta especifican completamente la estructura interna de un componente integrado, haciendo que la interfaz privada sea de vital importancia en la composición de componentes integrados. Los parámetros de entrada y salida "enlazados" pueden pertenecer tanto a instancias concretas de servicios web como a componentes integrados, tal como veremos en el proceso de creación de componente integrados en el capítulo 4.

La Figura 3.5 muestra los diferentes tipos de enlaces para un componente integrado. Los enlaces especifican reglas de transferencia, como por ejemplo copiar datos entre parámetros, pero también es posible la especificación de filtros mediante expresiones XPath[5]. Esto se debe a que normalmente no existe una correspondencia exacta entre la salida de un servicio y la entrada del siguiente, sino que en la mayoría de los casos es necesario establecer filtros que eliminen información adicional irrelevante para el flujo de datos de la composición. En función del tipo de parámetro involucrado en la transferencia de datos clasificamos los enlaces como sigue:

- Enlaces *input*. Este tipo de enlace describe las reglas de transferencia para los parámetros de entrada entre diferentes componentes integrados. Indican como fluye la información desde la composición, como el com-

[5] `http://www.w3.org/TR/xpath`

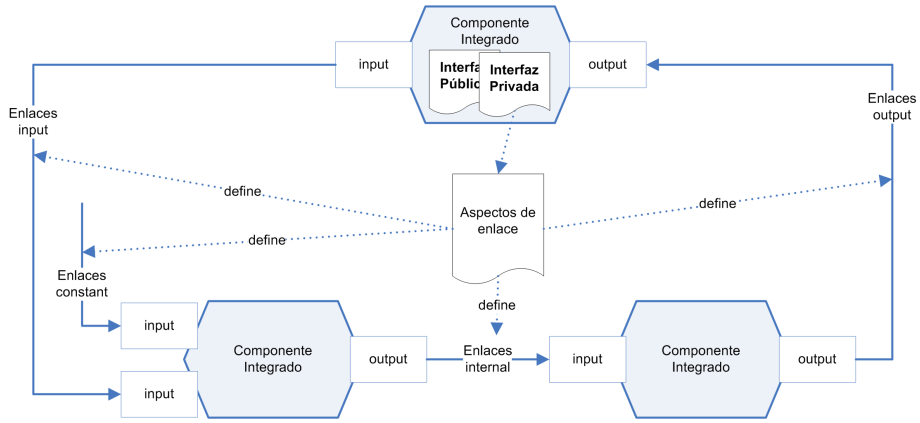


Figura 3.5: Aspectos de enlace de un componente integrado

ponente integrado situado en la parte superior de la Figura 3.5, hacia los componentes integrados contenidos en esa composición, que pueden ser tanto otros componentes integrados más simples como instancias de servicios web. La semántica de estos enlaces consiste en preparar las entradas de datos para la ejecución de los componentes contenidos (flechas hacia abajo en la parte izquierda de la Figura 3.5).

- Enlaces *output*. Definen las reglas de transferencia entre parámetros de salida de diferentes componentes integrados. En este caso, la información fluye desde los componentes contenidos en la composición hacia la propia composición. Es decir, una vez ejecutados los correspondientes componentes contenidos, el resultado es transferido a la composición para proseguir con la ejecución (flechas hacia arriba en la parte derecha de la Figura 3.5).

- Enlaces *internal*. Este tipo de enlace describe las reglas de transferencia entre componentes integrados al mismo nivel (flechas horizontales en la Figura 3.5). Normalmente envuelven transferencias entre parámetros de salida de un componente integrado y los parámetros de entrada del siguiente componente en ser ejecutado. También los enlaces *input* pueden reflejar reglas de transferencia entre parámetros de entrada o entre parámetros de salida, aunque suele ser menos habitual en escenarios reales.

- Enlaces *constant*. Los enlaces *constant* son útiles en los escenarios que se desee ofrecer la posibilidad de configurar una composición con ciertos valores constantes, bien porque resulta una configuración habitual o porque se desea encapsular ciertos valores a los usuarios finales. Obviamente, este tipo de enlaces se aplica únicamente a parámetros de entrada.

3.3. Influencia de los servicios web

Los servicios web proporcionan cierta funcionalidad remota accesible mediante estándares básicos basados en Internet [7]. Uno de estos estándares, WSDL, está encargado de la descripción de los interfaces de servicios web sin prestar atención a cómo está implementada la funcionalidad que ofrecen. El conjunto de etiquetas de la especificación WSDL se divide en dos partes lógicas. La parte abstracta describe las capacidades de un servicio, detallando las signaturas de las funciones que ofrece y sus parámetros de entrada y salida. La parte de implementación describe parámetros específicos como protocolos de transporte y puntos de comunicación del servicio, necesarios para la invocación de un servicio web. La Figura 3.6(a) muestra un extracto de la descripción WSDL para un servicio web gazetteer[6] basado en el Alexandria Digital Library (ADL) Gazetteer[7]. El servicio web `ADL Gazetteer` toma como entrada un topónimo, como por ejemplo un nombre de ciudad, y devuelve su localización como un par de coordenadas (x,y), comportamiento típico de un gazetteer.

Un componente integrado no incorpora únicamente la descripción WSDL utilizada en los servicios web. Un componente integrado comparte con los servicios web la idea de una representación descriptiva de una funcionalidad, ya que no es directamente ejecutable como un componente software, sino que describe una funcionalidad remota al igual que los servicios web. En este sentido, la composición y reutilización de componentes integrados sigue una arquitectura SOA ya que trabajamos con descripciones de interfaces de servicios y no con el código subyacente.

Los aspectos funcionales (parte de la interfaz pública) de un componente integrado son representados mediante una descripción WSDL equivalente. Sin embargo, tan solo es interesante la parte abstracta de la especificación WSDL, como por ejemplo las etiquetas `portType`, `message` y `part` que aparecen en la Figura 3.6(a). Esto es debido a que los componentes integrados pueden verse como composiciones virtuales que no son directamente ejecutables, por lo que no necesitan los detalles concretos especificado en la parte de implementación de una descripción WSDL.

Además de la descripción sintáctica, los aspectos funcionales también incorporan aspectos semánticos para anotar la funcionalidad de un componente integrado. Ya que WSDL únicamente proporciona descripciones sintácticas, necesitamos un mecanismo para enriquecer semánticamente dichas descripciones. La especificación WSDL-S [138] define un sencillo mecanismo para añadir semántica a los aspectos funcionales de un componente integrado basado en descripciones WSDL. En nuestra aproximación hemos utilizado algunas características de WSDL-S para anotar los nombres de funciones y sus parámetros con conceptos definidos en ontologías de dominio.

---

[6] Nomenclátor en castellano, aunque mantendremos el término anglosajón *gazetteer* a lo largo de esta tesis

[7] `http://middleware.alexandria.ucsb.edu/client/gaz/adl/index.jsp`

```
01 <?xml version='1.0' encoding='UTF-8'?>
02 <wsdl:definitions xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'/>
03   ...
04   <wsdl:message name='getCoordinatesResponse'>
05     <wsdl:part name='output' element='xsd1:ResponseType'/>
06   </wsdl:message>
07   <wsdl:message name='getCoordinatesRequest'>
08     <wsdl:part name='name' element='xsd1:RequestType'/>
09   </wsdl:message>
10   <wsdl:portType name='ADLGazClient'>
11     <wsdl:operation name='getCoordinates'/>
12   </wsdl:portType>
13   ...
14 </wsdl:definitions>
```

(a) Descripción WSDL

```
01 <?xml version='1.0' encoding='UTF-8'?>
02 <wsdl:definitions xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
03   xmlns:wssem='http://www.ibm.com/xmlns/WebServices/WSSemantics'/>
04   ...
05   <wsdl:message name='getCoordinatesResponse'>
06     <wsdl:part name='output' element='xsd1:ResponseType'
07       wssem:modelReference='ADLPoint'/>
08   </wsdl:message>
09   <wsdl:message name='getCoordinatesRequest'>
10     <wsdl:part name='name' element='xsd1:RequestType'
11       wssem:modelReference='ADLCityName'/>
12   </wsdl:message>
13   <wsdl:portType name='ADLGazClient'>
14     <wsdl:operation name='getCoordinates'
15       wssem:modelReference='LocSpat'/>
16   </wsdl:portType>
17 ...
18 </wsdl:definitions>
```

(b) Descripción WSDL-S

Figura 3.6: Descripción abstracta del servicio web `ADL Gazetteer`

La Figura 3.6(b) muestra la descripción WSDL-S equivalente a la descripción WSDL de la Figura 3.6(a). En concreto, las etiquetas `part` (líneas 06 y 10) de WSDL para los parámetros de entrada y salida de una función son anotadas mediante el atributo `modelReference` (líneas 07 y 11) de WSDL-S para describir su significado. Por ejemplo, el parámetro de entrada `name` (línea 10) se anota mediante el atributo `modelReference` al concepto `ADLCityName` (línea 11) que pertenece a una ontología de dominio. El prefijo `wssem` indica el espacio de nombres utilizado para la especificación WSDL-S (línea 03). El parámetro de salida `output` (línea 06) se anota del mismo modo. La operación `getCoordinates` (línea 14) hace referencia al concepto `LocSpat` (línea 15) definido en una ontología de geo-operaciones [85], definida como una operación

que devuelve un atributo de tipo espacial basado en una localización (i.e, un gazetteer).

3.4. Influencia de los sistemas de basados en componentes

Los sistemas basados en componentes tienen gran influencia en nuestro trabajo ya que nos proporcionan las bases para definir nuestro modelo de componente y la metodología de composición (véase Figura 3.1). En esta sección nos centramos únicamente en el modelo de componente, dejando la metodología de composición para el capítulo 4. La idea consiste en examinar cómo las características de un componente software derivan en el concepto de componente integrado formando el modelo de componente de nuestra aproximación.

El primer paso para comprender porque el concepto de componente integrado se convierte en modelo de componente, empieza por examinar que entendemos por componente software. Existen numerosas definiciones del término componente software, sin embargo a lo largo de este trabajo hemos adoptado la definición de Szyperski [142], por tratarse de la más extendida actualmente:

> *A software component is a unit of composition with contractually specified interface and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parts.*

La primera característica que podemos extraer de la definición anterior es que, por definición, un componente es el único ingrediente de un proceso de composición. Desde un principio podríamos haber seleccionado un servicio web como pieza básica de nuestra aproximación. Entonces la composición de servicios web simples produciría servicios web compuestos (o procesos), diferentes en naturaleza a los servicios web simples [7]. En consecuencia, nuestro modelo de componente constaría de dos tipos de componentes (servicios web simples y compuestos), violando la premisa inicial de que un servicio web simple fuera el único ingrediente del proceso de composición. Por este motivo, la definición inicial del modelo de componente de un sistema es un aspecto crítico porque simplifica enormemente la correspondiente metodología de composición. Así pues, el planteamiento de este trabajo fue considerar tanto los servicios web simples como los compuestos como una única entidad, denominada componente integrado, en vez de tratarlos como entidades distintas. La composición de componentes integrados produce como resultado un componente integrado, haciendo de la recursividad un mecanismo implícito y natural para construir composiciones complejas, reduciendo de este modo la complejidad de la metodología de composición. Para un usuario final, es totalmente transparente si un componente integrado esconde un simple servicio web o una cadena de servicios, ya que ambos están encapsulados bajo la misma interfaz.

Otro aspecto importante de la definición anterior es que los componentes deben ser unidades reemplazables. Esto significa que dados dos componentes con funcionalidad similar pueden ser intercambiables en la misma composición. Adoptamos esta característica en el concepto de componente integrado por medio de dos mecanismos. Por un lado, para que un componente integrado sea reemplazable debe evitar dependencias de datos externas con otros componentes integrados. De esta forma, es independiente de la composición de la cual forma parte, pudiendo ser reutilizado en otras composiciones que requieran su funcionalidad. La granularidad del componente integrado influye en el nivel de dependencia de datos. En general, la granularidad se define como el grado de refinamiento funcional. Por ejemplo, es más probable que la granularidad de los servicios web sea menor que en los componentes software. Un servicio web hará más en una sola operación (*coarse-grained*) que un componente en un método (*fine-grained*) simplemente por razones de rendimiento en el contexto de aplicación (Internet/Intranet). Resulta más prohibitivo juntar varios servicios web en un misma transacción que en el caso de los componentes software. Pues bien, básicamente atendemos a la regla de cuanto más pequeñas son las piezas de un sistema, más fácilmente son reutilizables porque éstas llevan a cabo funcionalidades limitadas sin depender de otras piezas para poder ejecutarse. Así pues, encontrar el nivel adecuado de granularidad de un componente integrado juega un papel importante a la hora de maximizar la reutilización de éstos. Este aspecto será de vital importancia durante la creación de componentes integrados (véase capítulo 4).

Por otra parte, la interfaz pública y privada de un componente integrado facilita la encapsulación porque la funcionalidad es únicamente visible a través de la interfaz pública, mientras que los aspectos estructurales y de enlace (la interfaz privada) permanecen ocultos a otros componentes integrados. De este modo, comprobar si dos componentes integrados ofrecen una funcionalidad similar consiste en comparar sus interfaces públicas sin tener en cuenta sus estructuras internas. Además, ya que la interfaz pública de un componente integrado está anotada semánticamente, esto proporciona un factor adicional de fiabilidad que comparar únicamente interfaces a un nivel sintáctico [89], ya que interfaces sintácticamente diferentes pueden realizar la misma funcionalidad (semánticamente idénticas). En este punto es interesante señalar que en esta tesis asumimos la utilización de ontologías de dominio compartidas. Normalmente tratamos con una única ontología para describir los conceptos con los que anotar los componentes integrados ya que nos ceñimos a un único contexto de actuación, facilitando la comparación de conceptos porque todos pertenecen a la misma ontología. Para nuestro caso disponemos de una geo-ontología[8] para la definición de conceptos en el contexto geográfico (por ejemplo los conceptos `Point` o `CityName`) así como para las definiciones de geo-operaciones (por ejemplo el concepto `LocSpat` para expresar la funcionalidad de un gazetteer).

[8] La ontología para el dominio geográfico o geo-ontología utilizada a lo largo de esta tesis ha sido desarrollada por Rob Lemmens como parte de su tesis doctoral [85]

3.5. Influencia de los patrones de workflow

Los patrones de un componente integrado están relacionados con el proceso de composición, en el sentido en que son uno de los principios que Szyperski argumenta para la definición de la metodología de composición [143]:

> *Composition process can be considered the application of a composition operator.*

Aunque la metodología de composición será detallada en el capítulo 4, en esta sección describimos la importancia de los patrones para el concepto de componente integrado, ya que los patrones serán los *operadores de composición* de los componentes integrados, especificados en los aspectos estructurales de un componente integrado. También analizaremos el proceso de selección de patrones específicos para la composición y reutilización de componentes integrados.

3.5.1. Patrones tradicionales

Como se ha mencionado en la primera parte de este capítulo, el proceso de composición planteado en este trabajo se basa en el análisis de los patrones propuesto por van der Aalst et al. [148]. Sin embargo, este conjunto de patrones fue planteado para establecer un marco unificado con el que comparar la funcionalidad de diferentes Workflow Management Systems (WfMS) existentes en el mercado, como por ejemplo WebSphere MQ Workflow[9] o Lotus Workflow[10]. Parece interesante mostrar, antes de comenzar el análisis de patrones, por qué hemos seleccionado estos patrones como base del proceso de composición de nuestra aproximación. Aunque *a priori* parece que se traten de dos áreas de investigación sin ninguna relación, la gran mayoría de las aproximaciones existentes encaminadas a la creación de un lenguaje de composición para servicios web tienen su origen en el área de los WfMS [5]. De hecho, algunos de estos lenguajes, como WSBPEL [105], utilizan una sintaxis propia de los WfMS como por ejemplo *proceso de negocio* o simplemente *proceso*, para designar una composición de servicios web. Además, recientemente se han publicado varios estudios comparativos entre los lenguajes de composición de servicios web y los patrones de workflow [140, 161, 150], subrayando el estrecho vínculo que existe entre ambas áreas.

Nuestro análisis de patrones toma como punto de partida los patrones tradicionales [148]. Sin embargo, a diferencia de otras aproximaciones, nuestro objetivo no consiste en la aplicación directa de dichos patrones en la composición de componentes integrados. Realmente, nuestro objetivo ha consistido en analizar dichos patrones en su nuevo contexto de aplicación (primero para los servicios web, y luego para los componentes integrados), eliminando patrones redundantes o sin funcionalidad aparente para este nuevo contexto. La idea

[9] `http://www-306.ibm.com/software/integration/wmqwf/`
[10] `http://www.lotus.com/workflow`

es conseguir un conjunto simplificado y unificado de patrones que permitan al usuario componer y reutilizar componentes integrados de forma sencilla y sin ambigüedades.

Antes de comenzar con el análisis de los patrones, describimos a continuación los criterios utilizados en el proceso de análisis. En primer lugar, el criterio de *simplicidad* ha sido uno de los objetivo de este análisis. Nuestra intención ha consistido siempre en minimizar el número de patrones redundantes cuando éstos son aplicados al contexto de servicios web, en contraposición por ejemplo a los patrones redundantes y alternativos existentes en WSBPEL [121]. De esta forma se simplifica tanto el diseño como la composición de componentes integrados desde el punto de vista del usuario. Además, los patrones derivados del análisis deberán ser acordes con las características de los componentes integrados, favoreciendo la creación y composición de componentes integrados *reutilizables*, *flexibles*, e *independientes* (sin dependencias de datos externas).

3.5.2. Análisis de patrones en el contexto de servicios web

El primer paso de nuestro análisis consiste en identificar un subconjunto de patrones relevantes en el contexto de los servicios web, tomando como punto de partida los 20 patrones listados en la Tabla 3.2. La metodología del análisis llevado a cabo consiste en describir brevemente algunos de estos patrones (véase capítulo 2) para analizar si son o no relevantes en el contexto de los servicios web, atendiendo a los criterios de selección expuestos anteriormente.

Tal como muestra la Tabla 3.2, estos 20 patrones están agrupados en cinco categorías según su funcionalidad. Los números entre paréntesis que aparecen a lo largo de esta sección hacen referencia al identificador de patrón en la primera columna de la Tabla 3.2.

Discusión: Basic control flow patterns

La primera categoría se denomina *basic control flow patterns* y engloba los patrones que capturan los aspectos elementales en un workflow. Por tratarse de los patrones más simples pero a su vez básicos, hemos considerado todos los patrones de esta categoría (1-5) como relevantes para los servicios web: *sequence* (1), *parallel split* (2), *exclusive choice* (4), y sus respectivos patrones homólogos que incorporan sincronización como *synchronization* (3) y *simple merge* (5).

Discusión: Advanced branching and synchronization patterns

En contraposición a los anteriores patrones, la categoría *advanced branching and synchronization patterns* (6-9) incluye patrones mucho más avanzados que incluso no son soportados completamente por algunos motores de workflow [76, 148] ni por la gran mayoría de lenguajes de composición de servicios web [140]. Los patrones *multi-choice* (6) y *synchronization merge* (7)

Id.	Patrón de workflow	Rel. SW	Rel. CI
Basic control flow patterns			
1	Sequence	√	√
2	Parallel split	√	√
3	Synchronization	√	√
4	Exclusive choice	√	√
5	Simple merge	√	√
Advanced branching and synchronization patterns			
6	Multi-choice	√	√
7	Synchronizing merge	√	√
8	Multi-merge	√	×
9	Discriminator	√	√
Structural patterns			
10	Arbitrary cycles	√	√
11	Implicit termination	×	×
Patterns involving multiple instances			
12	M.I.without synchronization	√	×
13	M.I.with a priori design time knowledge	√	×
14	M.I.with a priori runtime knowledge	×	×
15	M.I.without a priori runtime knowledge	×	×
State-based Patterns			
16	Deferred choice	√	×
17	Interleaved parallel routing	√	×
18	Milestone	×	×
Cancellation patterns			
19	Cancel activity	×	×
20	Cancel case	×	×

Tabla 3.2: Conjunto de patrones tradicionales y su relevancia para los contextos de servicios web y componentes integrados

son significativos para la composición de servicios web porque proporcionan la función lógica OR. El primero permite que, dado un determinado punto del workflow, varias ramas sean seleccionadas si se cumple cierta condición. El segundo es su complementario, se decir, permite que múltiples ramas converjan en un único punto de sincronización.

El siguiente patrón, *multi-merge* (8), también combina dos o más ramas pero sin sincronización. Supongamos por ejemplo que disponemos de dos servicios `gazetteer` que dado un nombre de una ciudad devuelven su correspondiente localización. También deseamos que, una vez ejecutados ambos servicios de forma paralela, se inicie un servicio de `web mapping` que devuelve un mapa centrado dada la localización anterior. Si modelamos este simple escenario mediante el patrón *multi-merge*, el servicio `web mapping` se ejecutaría dos veces, una por cada rama entrante (servicio `gazetteer`), ya que no existe sincronización. En principio, el patrón *multi-merge* es bastante común en escenarios reales donde el servicio que combina las ramas entrantes debe ejecutarse de forma independiente por cada uno de los servicios entrantes.

Por último, el patrón *discriminator* (9) resulta muy útil en el contexto de servicios web porque puede mejorar notablemente la flexibilidad de los servicios involucrados en una composición. La flexibilidad se define como la capacidad de servir el mejor de entre un conjunto de servicios web que ofrecen una funcionalidad similar. Por ejemplo, el mejor servicio puede ser uno que responde más rápidamente que los demás cuando se ejecuta. Supongamos otra vez que los dos servicios `gazetteer` son ejecutados en paralelo. Sin embargo, a diferencia del patrón *multi-merge*, el servicio `web mapping` se ejecuta una única vez justo en el instante en que recibe el resultado de cualquiera de los dos `gazetteer`, ignorando la respuesta del servicio `gazetteer` más lento. En este caso, existen dos servicios alternativos con la misma funcionalidad y aplicados al mismo contexto, retrasando la elección final del servicio hasta el momento de la invocación. Sin embargo, habrá que tener en cuenta que la descripción de composiciones flexibles implica la especificación de las mínimas restricciones sintácticas y semánticas, para garantizar lo máximo posible servicios débilmente acoplados (*loosly coupled*) que sean fácilmente intercambiables [69].

Discusión: Structural patterns

Los patrones *arbitrary cycles* (10) y *implicit termination* (11) forman el grupo de *structural patterns*. Como ocurre en los procesos de workflow tradicionales, los bucles son constructores básicos para modelar composiciones de servicios web en las que se requiera que ciertos servicios deban ser ejecutados repetidas veces. El patrón *implicit termination* (11) termina la ejecución de workflow simplemente porque no hay nada más que hacer. Desde el punto de vista de los servicios web, no resulta necesario añadir explícitamente a una composición un servicio de terminación.

Discusión: Patterns involving multiple instances

El siguiente grupo de patrones comparten la característica de poseer múltiples instancias. Desde un punto de vista práctico significa que una actividad de un workflow puede tener más de una instancia activa y ejecutándose al mismo tiempo. Los dos primeros patrones (12-13) son interesantes en el contexto de los servicios web. El patrón 12 implica que varias instancias de una actividad están activas sin ninguna sincronización entre ellas. Por ejemplo, deseamos obtener la localización de una lista de ciudades. Aplicando el patrón 12, por cada entrada de la lista se crea una instancia del servicio `gazetteer`. A medida que cada instancia termina, se inicia el servicio `web mapping` sin que exista sincronización entre las instancias del servicio `gazetteer` activas. El patrón 13 es muy similar al anterior pero esta vez si se aplica sincronización a las diferentes instancias activas de una actividad. Ahora, todas las instancias del servicio `gazetteer` deben finalizar antes de iniciar el servicio `web mapping`. Respecto a los restantes patrones para múltiples instancias (14-15), consideramos que no son relevantes para la composición de servicios web porque el número de instancias es desconocido en tiempo de diseño.

Discusión: State-based patterns

Típicamente, las actividades en escenarios de workflow tradicionales permanecen más tiempo en estado de espera que en proceso [76]. Esto significa que las actividades son activadas cuando reciben ciertos mensajes de activación mediante un sistema de notificación de eventos. Los siguientes patrones describen situaciones en las cuales el estado de una actividad cambia en función de un evento externo a la propia actividad.

El patrón *deferred choice* (16) es prácticamente idéntico al patrón *exclusive choice* (4) exceptuando en la naturaleza de la condición. En ambos patrones una de las posibles ramas es activada dependiendo de una condición. En el patrón *exclusive choice* la condición viene dada por decisiones sobre datos de la propia actividad, mientras que para el patrón *deferred choice* la condición se basa en decisiones sobre datos externos. Este patrón resulta atractivo para los servicios web ya que este comportamiento es bastante común en escenarios reales.

El siguiente patrón, *interleaved parallel routing* (17), define la ejecución de una secuencia de actividades sin ningún orden establecido. Aunque en el contexto de los servicios web el usuario suele prefijar una secuencia en tiempo de diseño, suponemos interesante este patrón.

Por último, aunque el patrón *milestone* (18) tiene gran importancia durante la ejecución de un workflow, no tiene un impacto relevante para la composición de servicios web. Además, aplicando el criterio de simplicidad, el patrón *milestone* puede ser modelado mediante la combinación de patrones más básicos [101] como *sequence* (1), *parallel split* (2), *choice* (4, 6), y *arbitrary cycles* (10).

Discusión: Cancellation patterns

Los patrones de cancelación permiten cancelar la ejecución de un actividad o incluso de todo un workflow. En el contexto de los servicios web, no consideramos necesario disponer de una patrón explícito de cancelación. Como en el caso del patrón *implicit termination* (11), esta decisión se basa en el criterio de simplicidad, ya que no aportan una funcionalidad significativa para la composición de servicios web.

3.5.3. Análisis de patrones en el contexto de componentes integrados

El conjunto de patrones considerados para la composición de servicios web son el punto de partida para la definición de los patrones en el contexto de los componentes integrados. Por lo tanto, tenemos en cuenta los patrones marcados como $\surd$ en la tercera columna de la Tabla 3.2.

Discusión: Patrones secuenciales

Comenzamos el análisis con el caso más sencillo, los patrones de secuencia. Consideramos únicamente secuencias ordenadas (1) de componentes integrados prefijadas en tiempo de diseño. El patrón de secuencias desordenadas (17) establece el orden en tiempo de ejecución [148]. Ya que nuestro trabajo se centra en tiempo de diseño, prescindimos del patrón *interleaved parallel routing* considerando el orden de los componentes integrados conocido en tiempo de diseño. Por lo tanto, una secuencia de componentes integrados siempre se definirá mediante el patrón *sequence* (1).

Discusión: Patrones paralelismo

Entre los patrones involucrados en paralelismo (2-8, 16), hemos seleccionado el conjunto mínimo de patrones con los que se definen las funciones lógicas AND, XOR y OR. Para cada función lógica hemos considerado el patrón de bifurcación (*split*) junto con el correspondiente patrón de sincronización (*join*). La función lógica AND incluye el patrón *parallel split* (2) que representa una bifurcación junto con el patrón *synchronization* (3). El patrón XOR queda definido por los patrones *exclusive choice* (4) y *simple merge* (5), mientras que la pareja de patrones *multi-choice* (6) y *synchronizing merge* (7) representa un OR lógico.

Los anteriores seis patrones suelen estar presentes en cualquier workflow y son implementados por la mayoría de los WfMS y lenguajes de composición de servicios web [140]. Pero la mayoría de estos sistemas y lenguajes ofrecen únicamente las funciones AND y XOR, simplificando la función OR (varias ramas que pueden ser activadas) a una función XOR (únicamente una rama es activada). En el contexto de los componentes integrados hemos preferido incluir

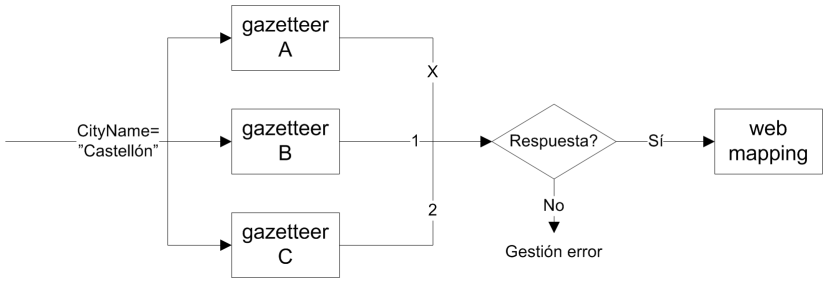
teóricamente la función OR para aumentar la flexibilidad de una composición, como será discutido en la siguiente sección 3.5.4, aunque en la práctica la función OR será tratada como una XOR por la falta de soporte en los actuales lenguajes de composición [140].

Los dos patrones restantes para el paralelismo (8,16) no son apropiados en el contexto de los componentes integrados. El patrón *multi-merge* (8) puede ser representado como una combinación de los patrones *parallel split* (2) y *synchronization* (3), manteniendo diferentes ejecuciones en paralelo sin sincronización [71]. Por lo tanto, no incluimos este patrón ya que uno de nuestros criterios de selección es justo evitar la existencia de patrones alternativos y redundantes. Tampoco consideramos el patrón *deferred choice* (16) para los componentes integrados ya que la condición para la bifurcación depende de un valor externo al propio componente integrado. Debido a esto, este patrón no es válido ya que va en contra de uno de los principios básicos de los componentes integrados: evitar dependencias de datos externas con otros componentes con la finalidad de incrementar el grado de independencia y reutilización.

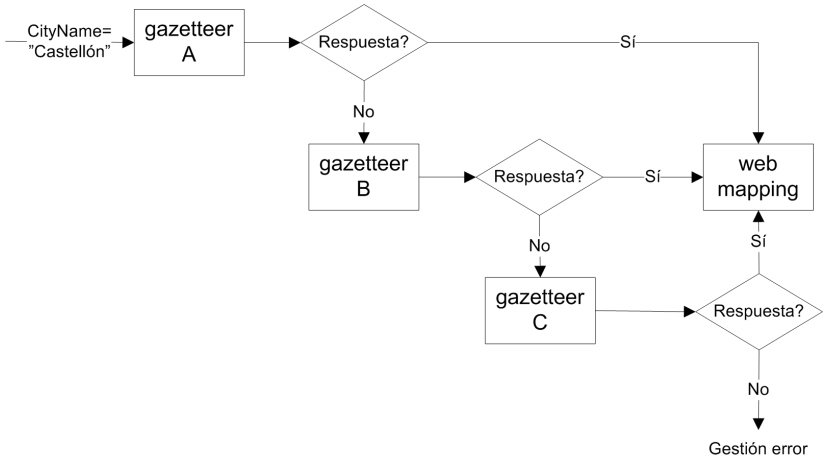
Discusión: Discriminator

En la sección 3.5.2 señalábamos la importancia del patrón *discriminator* (9) en el contexto de los servicios web. También lo es para los componentes integrados ya que facilita la flexibilidad de las composiciones de componentes integrados. La Figura 3.7(a) muestra el significado original del patrón *discriminator* expuesto en [148]. Tres servicios `gazetteer` son invocados paralelamente con el mismo parámetro de entrada (`CityName=‘‘Castellón’’`). Supongamos que el `gazetteerA` no está disponible por caída del servidor. De los otros dos, el `gazetteerB` es el primero en responder mientras que el `gazetteerC` lo hace en segundo lugar. El patrón *discriminator* toma únicamente el resultado del `gazetteerB` para continuar con el workflow ejecutando el servicio `web mapping`. ¿Como influye este comportamiento en la flexibilidad?. En tiempo de diseño no sabemos cual de los servicios (`gazetteerA`, `gazetteerB`, `gazetteerC`) será ejecutado completamente, tan solo hemos proporcionado un conjunto de servicios candidatos que ofrecen la funcionalidad de un `gazetteer` requerida en la composición. La ventaja de este patrón es que nos permite diseñar composiciones de servicios web mucho más flexibles porque cada actividad de un workflow no está ligada a un servicio web concreto, sino que está asociada a un conjunto de servicios que ofrecen la funcionalidad que requiere dicha actividad.

Desafortunadamente, los actuales lenguajes de composición de servicios web no implementan el comportamiento del patrón *discriminator* original [140] (véase Figura 3.7(a)). Para adoptarlo al contexto de los componentes integrados, hemos derivado una versión secuencial del patrón *discriminator*. El significado original permanece invariable, es decir, únicamente el resultado de un servicio es considerado, pero sin embargo los servicios son ejecutados de forma secuencial debido a que los actuales lenguajes de composición no implementan



(a) Discriminador original



(b) Discriminador secuencial adaptado para los componentes integrados

Figura 3.7: Representación gráfica del patrón *discriminator*

este patrón, ni siquiera en la versión secuencial. La Figura 3.7(b) representa la versión secuencial adoptada en nuestra aproximación. La ejecución comienza con el `gazetteerA`. Si no se produce error la composición continúa y los restantes servicios ni siquiera son ejecutados. Si el `gazetteerA` falla, entonces se intenta la ejecución del siguiente `gazetteer` disponible (`gazetteerB`). A diferencia del patrón *discriminator* original, los servicios son procesados secuencialmente y además, puede darse el caso en que no todos son ejecutados. Aunque hayamos perdido la capacidad de procesamiento en paralelo, mantenemos intacta la esencia del patrón *discriminator* y con ello la capacidad de generar composiciones flexibles.

Discusión: Arbitrary cycle

De forma similar al patrón anterior, derivamos un patrón para bucles más simple que el original. El patrón *arbitrary cycle* (10) define bucles generales con

un número arbitrario de entradas y salidas, similar a una instrucción GOTO en programación tradicional. Como este patrón tampoco es soportado por la mayoría de los lenguajes de composición [140] y, atendiendo al criterio de simplicidad, asumimos una versión simple del patrón *arbitrary cycle* que consta únicamente de un punto de entrada y uno de salida, equiparable a instrucciones de programación WHILE. Este nuevo patrón sigue manteniendo el significado del patrón original pero sin embargo ya es directamente soportado por los lenguajes de composición existentes.

Discusión: Patrones múltiples instancias

Finalmente, los patrones de múltiples instancias (12, 13) que hemos tenido en cuenta para los servicios web, no son considerados para el contexto de los componentes integrados. Hemos aplicado el mismo razonamiento que para prescindir del patrón *milestone* (18), es decir, ambos patrones pueden ser modelados mediante una combinación de patrones más simples [101], ya seleccionados previamente durante este análisis.

3.5.4. Patrones de selección y de composición

La última parte de este análisis consiste en determinar los patrones que finalmente serán utilizados en nuestra aproximación. Hemos clasificado los patrones relevantes para los componentes integrados (cuarta columna, Tabla 3.2) en dos categorías, *selección* y *composición*, según el contexto en que son aplicados. Los patrones de *selección* se aplicarán durante la creación de componentes integrados a partir de instancias de servicios web. Los patrones de *composición* se aplicarán durante la composición de componentes integrados a partir de otros componentes integrados ya creados. Ambas categorías de patrones comparten la idea original de ser operadores de composición, aunque presentan matices diferentes. La Tabla 3.3 muestra los ocho patrones definitivos, dos patrones de *selección* (SP1-2) y seis patrones de *composición* (PC1-6), derivados a partir de los patrones seleccionados en el contexto de los componente integrados (véase sección 3.5.3).

Patrones de composición

Debido a que el objetivo de los patrones es servir como operadores de composición de componentes integrados, la combinación recursiva de componentes integrados produce una estructura jerárquica en forma de árbol. Como muestra la Figura 3.8, cada componente integrado genera una estructura en árbol en la cual los patrones conectan los nodos padres (componentes integrados) con los nodos hijos directos (bien componentes integrados o instancias de servicios web). Por ejemplo, el componente integrado raíz de la Figura 3.8 está compuesto de tres componentes integrados. En los atributos estructurales del componente integrado raíz se define el patrón de composición que indica cómo se

Id.	Patrón CI	Combinación de patrones
Patrones de selección		
PS1	AND-DISC	Parallel split (2) + sequential discriminator (9)
PS2	OR-DISC	Multi-choice (6) + sequential discriminator (9)
Patrones de composición		
PC1	SEQ	Sequence (1)
PC2	AND	Parallel split (2) + synchronization (3)
PC3	XOR	Exclusive choice (4) + simple merge (5)
PC4	OR	Multi-choice (6) + synchronization merge (7)
PC5	LOOP-COND	Conditional loop (10)
PC6	LOOP-ITER	Iterative loop (10)

Tabla 3.3: Patrones de selección y de composición

estructuran (secuencia o paralelo, por ejemplo) los tres componentes integrados contenidos, relacionando el nodo padre con sus nodos hijo directos. Para invocar la funcionalidad del componente integrado raíz, será necesario recorrer la estructura en árbol mediante un algoritmo en profundidad donde cada subárbol se recorre completamente ante de pasar al siguiente. Por ejemplo, todos los servicios web del `componente integrado`$_1$ son visitados antes que el `componente integrado`$_2$. Esto implica que los nodos hijos siempre son visitados antes que su correspondiente nodo padre. Por esta razón, para los patrones que especifican las funciones lógicas (AND, XOR, OR) debemos considerar parejas de patrones donde la condición de bifurcación (*split*) tiene lugar en los nodos hijo (son visitados antes) mientras que la condición de sincronización (*join*) tiene lugar en el nodo padre (es visitado posteriormente), tal como ilustra la Figura 3.8.

Según la Tabla 3.2, tenemos tres patrones que indican bifurcación (2, 4 y 6) que pueden combinarse con los tres patrones de sincronización (3, 5 y 7). Sin embargo, únicamente algunas de las posibles combinaciones encajan. Consideramos que un patrón de bifurcación y uno de sincronización son compatibles si ambos expresan la misma función lógica (AND, XOR, OR). La Tabla 3.3 muestras las tres combinaciones posibles, una para cada función lógica (PC2-4). Formalmente, si definimos CI como el componente integrado raíz y CI_i como cada uno de los componentes integrados contenidos en la composición CI, los patrones de composición PC2-4 son funciones que se aplican al conjunto $CI_1, CI_2, \ldots, CI_n$, siendo *cond* la condición de bifurcación:

$$(PC2) \qquad CI = AND(CI_1, CI_2, \ldots, CI_n)$$

$$(PC3) \qquad CI = XOR(CI_1, CI_2, \ldots, CI_n, cond)$$

$$(PC4) \qquad CI = OR(CI_1, CI_2, \ldots, CI_n, cond)$$

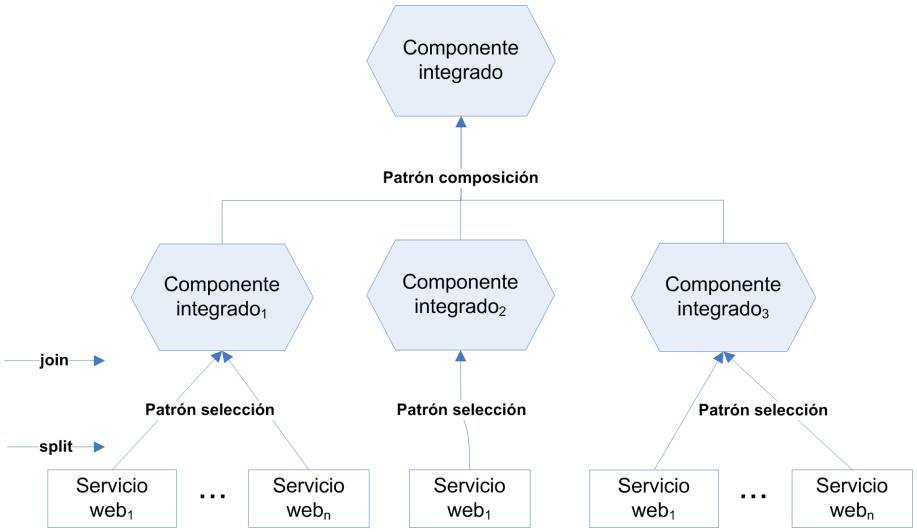


Figura 3.8: Estructura jerárquica de componentes integrados

La especificación del patrón secuencial es inmediata ya que no necesita ningún tratamiento especial. Simplemente, el patrón SEQ (PC1) de la Tabla 3.3 representa la secuencia ordenada de componentes integrados formalmente expresada como:

$$(PC1) \qquad CI = SEQ(CI_1, CI_2, \ldots, CI_n)$$

Dos tipos de patrones para bucles simples son derivados a partir del patrón *arbitrary cycle* (10). El patrón LOOP-COND (PC5) repite un componente integrado CI_1 hasta que cierta condición *cond* es evaluada cierta. Es importante señalar que esta condición tiene que ser especificada en términos de datos internos descritos en el propio componente integrado, preservando su autonomía e independencia. Además asumimos que el usuario, en el momento de diseñar la composición, introduce valores correctos para la condición con el fin de evitar bucles infinitos. Formalmente, el patrón LOOP-COND queda expresado como:

$$(PC5) \qquad CI = LOOP - COND(CI_1, cond)$$

Por otra parte, derivamos el patrón LOOP-ITER (PC6) que representa un bucle que repite n veces, especificado en tiempo de diseño, un componente integrado CI_1 dado:

$$(PC6) \qquad CI = LOOP - ITER(CI_1, n)$$

En definitiva, los patrones de *composición* permiten crear nuevos componentes integrados a partir de componentes integrados ya creados ($CI \longrightarrow CI$),

destacando la idea de que el concepto de componente integrado es la unidad básica reutilizable para construir composiciones.

Patrones de selección

Los patrones de *selección* son utilizados para la creación de componentes integrados. Su nombre se debe a que se basan en el patrón discriminador secuencial, que nos permite *seleccionar* el resultado de un servicio web a partir de un conjunto de servicios web candidatos. La Figura 3.8 muestra como el `componente integrado`$_1$ agrupa un conjunto de servicios web (`servicio web`$_1$...`servicio web`$_n$) similar funcionalmente, de los cuales tan solo el resultado obtenido de uno de ellos será considerado en tiempo de ejecución. Al igual que los patrones de composición PC2-4, necesitamos crear parejas de patrones que combinen la condición de bifurcación y de sincronización. El patrón discriminador secuencial actúa como punto de sincronización, ya que tiene lugar en el componente integrado (nodo padre). Para la condición de bifurcación hemos seleccionado los patrones *parallel split* (2) y *multi-choice* (6) asociados al conjunto de servicios web (nodos hijo). Así obtenemos dos patrones de selección, AND-DISC (PS1) y OR-DISC (PS2). El primero considera *a priori* todos los servicios web disponibles, es decir, cualquiera de los servicios web puede ser finalmente seleccionado. El segundo, en cambio, especifica cierta condición inicial que restringe el conjunto inicial de servicios web. Por ejemplo, si disponemos de servicios web candidatos sin coste y de pago, el usuario puede estar interesado únicamente en servicios gratis. El patrón OR-DISC seleccionará el primer servicio web que finalice de entre el subconjunto de servicios sin coste, ignorando desde el primer momento los servicios de pago. Por lo tanto, dado un conjunto de servicios web $SW_1, SW_2, \ldots, SW_n$, los patrones de selección se definen como:

$$(PS1) \qquad CI = AND - DISC(SW_1, SW_2, \ldots, SW_n)$$

$$(PS2) \qquad CI = OR - DISC(SW_1, SW_2, \ldots, SW_n, cond)$$

Resumiendo, los patrones de *selección* permiten crear componentes integrados a partir de servicios web ($SW \longrightarrow CI$), facilitando la flexibilidad de disponer de servicios web alternativos para un mismo componente integrado.

3.6. Resumen

Con la introducción del concepto de componente integrado, hemos intentado mantener lo más simple posible una aproximación para la composición de servicios web que facilite la reutilización y flexibilidad de composiciones. Hemos visto como un componente integrado se considera la pieza básica reutilizable, cogiendo para ello las características más interesantes de los sistemas basados en componentes. A la hora de proporcionar componentes integrados flexibles,

que permiten seleccionar un servicio web de entre un conjunto candidato funcionalmente similar, nos hemos basado en los patrones de workflow tradicionales. Gran parte de este capítulo se ha dedicado a realizar un exhaustivo análisis de estos patrones con el fin de encontrar un conjunto simplificado y unificado de patrones de composición en el contexto de los componentes integrados. Finalmente, los componentes integrados también adoptan la misma especificación de los servicios web para describir su interfaz funcional, haciendo indistinguible externamente (para un usuario o aplicación cliente) un componente integrado de un servicio web.

Capítulo 4

Metodología de composición

Este capítulo, el cual presenta la metodología de composición, enlaza con la descripción de componente integrado descrita en el capítulo anterior. La metodología propuesta consiste en un conjunto de procesos conceptuales para la creación, composición, reutilización y transformación de componentes integrados en procesos de workflow ejecutables. También muestra como el concepto de componente integrado junto con la metodología de composición se ajusta perfectamente a una arquitectura dividida en niveles de abstracción que favorecerá la creación de composiciones reutilizables y flexibles. El capítulo 5 presentará un herramienta software para el diseño de componentes integrados.

4.1. Introducción

Los lenguajes de composición de servicios web proporcionan a los usuarios aplicaciones basadas en procesos de workflow cuyas actividades son descritas como servicios web [105]. Típicamente este tipo de aplicaciones sigue una arquitectura de dos niveles, el nivel de aplicaciones, donde se encuentran los procesos, y el nivel de servicios concretos, que se corresponde con el espacio de servicios web disponible. Este tipo de lenguajes presentan limitaciones con respecto a la reutilización y flexibilidad de los procesos generados. Ante procesos que involucren muchos servicios web, no se presta atención a la reutilización de los subprocesos internos que generalmente pueden ser de utilidad para otros usuarios. Los subprocesos son dependientes del proceso del cual forman parte y únicamente existen dentro de ese proceso. Por otra parte, estos procesos son poco flexibles porque cada actividad del proceso se corresponde con un determinado servicio web concreto en tiempo de diseño. Si el servicio web no

está disponible en tiempo de ejecución (problemas de red o caída del servidor), la actividad asociada falla porque no dispone de servicios web alternativos para llevar a cabo dicha actividad, provocando el fallo completo del proceso.

Nuestra aproximación basada en componentes integrados (capítulo 3) junto con la metodología de composición (capítulo 4) genera procesos de workflow reutilizables y flexibles. Son reutilizables porque los subprocesos internos que forman un proceso son considerados a su vez procesos independientes que no dependen del proceso del que forman parte. Son flexibles porque cada actividad del proceso no está ligada a un servicio web sino a un conjunto de servicios web candidatos, retrasando la decisión de seleccionar un servicio web concreto lo máximo posible, idealmente hasta el momento de ejecución del proceso.

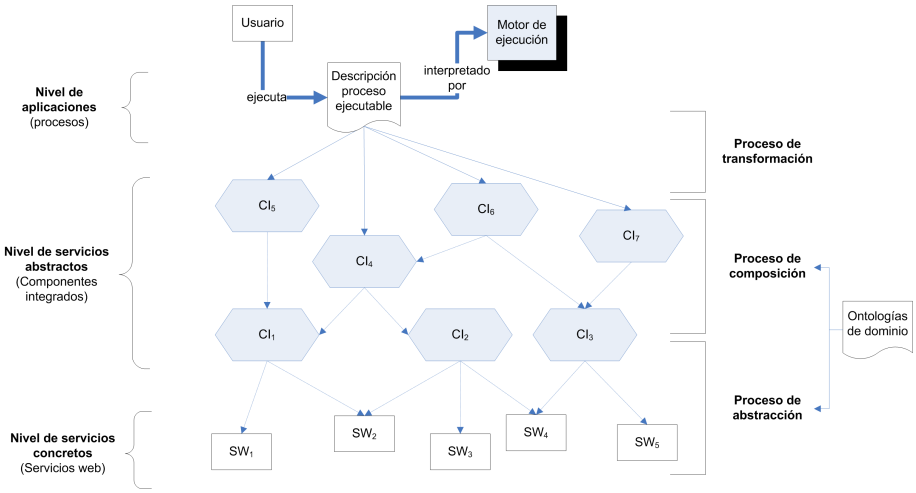


Figura 4.1: Arquitectura por niveles de abstracción

Para ello, nos basamos en las ideas de abstracción y descomposición como herramientas para simplificar y facilitar la estructura de sistemas de software complejos [156, 164]. Hemos añadido un nivel de abstracción adicional entre el nivel de aplicaciones (procesos) y el de servicios concretos (servicios web) que se corresponde con el nivel de servicios abstractos o componentes integrados, tal como muestra la Figura 4.1. El principio fundamental de esta arquitectura en niveles de abstracción es que existe una relación entre niveles consecutivos, donde componentes integrados de un nivel dado son definidos (o implementados) por los componentes integrados del nivel inmediatamente inferior. Es decir, un componente integrado de un nivel se *descompone* en los componentes integrados más simples del nivel inferior. Diremos que existe una relación de *descomposición* o *implementación* entre niveles adyacentes, tal como propone [156]. Tomemos por ejemplo el componente integrado $\texttt{CI}_4$ de la Figura 4.1. Este componente está implementado por los componentes integrados de nivel

inferior $\texttt{CI}_1$ y $\texttt{CI}_2$. A su vez, $\texttt{CI}_1$ se define como la composición de los servicios web del nivel inferior $\texttt{SW}_1$ y $\texttt{SW}_2$, y $\texttt{CI}_2$ como la composición de los servicios web $\texttt{SW}_2$, $\texttt{SW}_3$ y $\texttt{SW}_4$. Si suponemos que $\texttt{CI}_4$ define un proceso ejecutable, todos los componentes integrados de nivel inferior ($\texttt{CI}_1$ y $\texttt{CI}_2$) son composiciones independientes que pueden reutilizarse en otros componentes integrados de nivel superior, como es el caso de $\texttt{CI}_1$ por $\texttt{CI}_5$.

La relación de descomposición entre niveles adyacentes impide que se pueda hacer referencia entre niveles no consecutivos. Resulta imposible acceder a los servicios web involucrados en el componente integrado $\texttt{CI}_4$ sin antes pasar por los componentes integrados intermedios $\texttt{CI}_1$ y $\texttt{CI}_2$. La Figura 4.1 muestra como, debido a dicha relación entre niveles, cualquier componente integrado produce una estructura jerárquica en árbol donde sus nodos hijo directos son los componentes integrados del nivel inmediatamente inferior. De esta forma, resulta muy fácil averiguar la funcionalidad completa de un componente integrado dado que los nodos hijo siempre son la descomposición funcional del nodo padre.

En el capítulo 3 definíamos los componentes integrados como las unidades básicas reutilizables para producir composiciones de servicios web reutilizables y flexibles. En este capítulo nos centramos en la metodología de composición que, utilizando también ontologías de dominio como recurso adicional (véase Figura 4.1), proporciona un conjunto de procesos y mecanismos para la creación, composición y transformación de componentes integrados en procesos ejecutables. Atendiendo a su funcionalidad, hemos dividido la metodología de composición en tres grandes procesos conceptuales que serán descritos detalladamente a lo largo de este capítulo:

- El *proceso de abstracción* establece los mecanismos para la creación de componentes integrados a partir de servicios web del nivel de servicios concretos. Teniendo un diccionario a mano, el término *abstracción* se define como la capacidad mental que posee todo ser humano para poder comprender la esencia o concepto de un objeto. En nuestro contexto, el proceso de abstracción se entiende como la capacidad de agrupar un conjunto de servicios web con funcionalidad similar en un mismo componente integrado que representa a dicho conjunto. Salvando las diferencias entre ambos contextos, un componente integrado actúa de esencia o concepto de los servicios web que representan los objetos tangibles. Así, la funcionalidad de un componente integrado es implementada por cualquiera de los servicios web que agrupa, añadiendo cierta flexibilidad en la selección final del servicio web.

- El *proceso de composición* describe los mecanismos que permiten la creación de componentes integrados del nivel superior a partir de la combinación de otros componentes integrados del nivel inmediatamente inferior. La funcionalidad del nuevo componente integrado se descompone en la funcionalidad de los componentes integrados reutilizados. Claramente, la

reutilización *per se* tiene lugar durante el proceso de composición, en el sentido en que nos basamos en trabajo ya hecho por nosotros u otros usuarios –componentes integrados ya disponibles– para seguir avanzando en la creación de componentes integrados con nueva funcionalidad.

- El *proceso de transformación* es el encargado de transformar la descripción de un componente integrado en una representación equivalente que sea directamente ejecutable por un motor de ejecución. Como muestra la Figura 4.1, los componentes integrados se sitúan en el nivel de servicios abstractos, es decir, no son directamente ejecutables porque en realidad contienen descripciones abstractas (sin detalles de implementación). Por lo tanto, este proceso será necesario únicamente cuando un usuario desee invocar la funcionalidad descrita por un componente integrado.

4.2. Proceso de abstracción

El proceso de abstracción consiste principalmente en la creación de componentes integrados agrupando servicios web que comparten cierta funcionalidad. El componente integrado *abstrae* las diferencias sintácticas de las operaciones que ofrecen los servicios web bajo una misma función que además está descrita semánticamente. De esta forma, el proceso de abstracción produce componentes integrados flexibles, ya que para una funcionalidad dada descrita se dispone de un conjunto de servicios web para llevarla a cabo. La idea, pues, es separar en tiempo de diseño (a diferencia de WSBPEL) la interfaz de la operación que se ofrece del servicio concreto que puede implementar dicha operación.

La Figura 4.2 muestra un diagrama de actividad UML para el proceso de abstracción. En primer lugar el usuario realiza una búsqueda en el repositorio de servicios web (que también contiene descripciones funcionales de los componentes integrados, como veremos en la sección 4.2.3) para descubrir los servicios web que cumplen sus necesidades. Durante este proceso supondremos la existencia de servicios web (o componente integrados) que ya satisfacen las demandas de los usuarios. Si no fuera el caso, los proveedores de servicios son los encargados, como paso previo al proceso de abstracción, de la implementación de los servicios web requeridos y de su posterior publicación en repositorios de servicios web accesibles a los usuarios.

El proceso de abstracción tiene sentido cuando no existe aún un componente integrado con la funcionalidad requerida. Obviamente, si ya existe un componente integrado no se debería duplicar, sino que se debe (re)utilizar directamente. Tal como muestra la Figura 4.2, el proceso de abstracción consta de las siguientes fases: una vez encontrado el conjunto candidato de servicios web (sección 4.2.1), procedemos con la creación del componente integrado (sección 4.2.2) y finalmente, registramos el nuevo componente integrado en el repositorio para que pueda ser reutilizado por otros usuarios (sección 4.2.3).

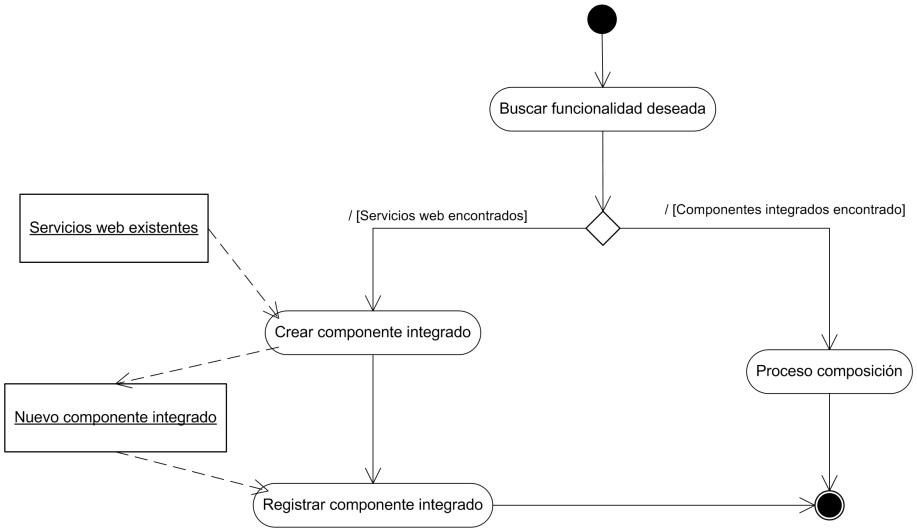


Figura 4.2: Proceso de abstracción

4.2.1. Selección de servicios web

En general la composición de servicios web puede desglosarse a su vez en tres tareas básicas [141]: la primera consiste en confeccionar un plan de las actividades necesarias para resolver el problema; la segunda tarea es la localización de los servicios web que se ajusten a las actividades del plan; y la tercera tarea consiste en realizar la composición de estos servicios web de forma que interactúen entre ellos de forma adecuada.

La tarea de la localización de servicios web está fuera del alcance de esta tesis. Suponemos que ya tenemos disponibles ciertos servicios web que han sido previamente localizados. La investigación llevada a cabo en esta tesis se ha centrado en la tercera tarea, en realizar de forma más eficiente la composición de servicios web debido a que se reutilizan partes creadas previamente. Recientemente, hemos aportado una posible solución mixta para resolver el problema completo de la composición de servicios web integrando dos aproximaciones [86, 56]: una para la localización de servicios a partir de un plan o consulta semántica [85], junto con nuestra aproximación basada en componentes integrados para la composición y reutilización de servicios web.

4.2.2. Creación de componentes integrados

La creación de un componente integrado es la parte principal del proceso de abstracción y consiste básicamente en la especificación de todos los aspectos que definen a un componente integrado: descriptivos, funcionales, estructurales, y de enlace. La Figura 4.3 muestra en detalle los diferentes actores que

intervienen en la creación de un componente integrado. El proceso de creación toma como entradas las descripciones funcionales de los servicios web (nivel de servicios concretos) y las ontologías de dominio, para producir como salida una descripción ICML para el nuevo componente integrado (nivel de servicios abstractos).

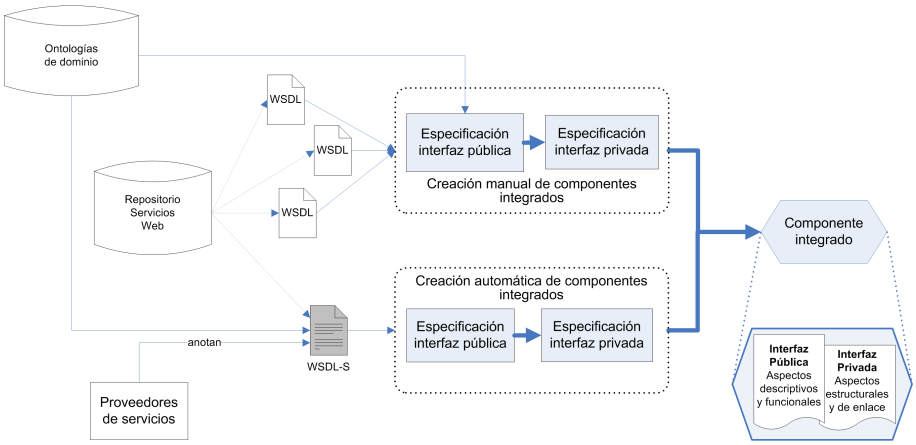


Figura 4.3: Creación de un componente integrado durante el proceso de abstracción

En realidad, tal como muestra la Figura 4.3, existen dos mecanismos diferentes para producir un nuevo componente integrado en función de si las descripciones de los servicios web iniciales incorporan o no semántica. Utilizamos la *creación manual* si los servicios web tiene únicamente una descripción sintáctica en formato WSDL. El usuario es el encargado de anotar *manualmente* la descripción funcional del componente integrado a partir de las ontologías de dominio. Sin embargo, en la *creación automática* la descripción funcional del servicio web está descrita en WSDL-S [138], por lo que ya incorpora semántica. Cada vez resulta más común la segunda opción en dominios restringidos, donde los proveedores de servicios inicialmente ya describen sus servicios semánticamente mediante ontologías de dominio previamente acordadas [127, 50]. Es importante destacar que la creación automática puede llevarse a cabo siempre que se utilice la misma ontología tanto para describir los servicios web como los componentes integrados. Es un requisito de nuestra aproximación que los componentes integrados creados en diferentes niveles de abstracción utilicen el mismo conjunto de conceptos. En caso de tratarse de ontologías diferentes, sería necesario realizar previamente un alineamiento e integración entre ontologías [102, 49], área que escapa al ámbito de esta tesis.

Creación manual

En la creación manual, por definición, el usuario es el encargado de especificar los aspectos del nuevo componente integrado. Como paso previo, el usuario tiene acceso tanto a las descripciones WSDL de los servicios web que formarán parte del nuevo componente integrado como a las ontologías de dominio, codificadas en OWL. Los aspectos del nuevo componente integrado serán declarados en el siguiente orden: primero los descriptivos y funcionales, y por último los estructurales y de enlace.

Los aspectos descriptivos especifican información textual, de ayuda para al usuario, relacionada con el contexto del nuevo componente integrado, como por ejemplo la descripción de la funcionalidad que ofrece o el punto de acceso (URL) a la descripción funcional (WSDL-S).

```
01 <FunctionalitySection>
02   <Operation>
03     <Name>getCoordinates</Name>
04     <OntologyResource>LocSpat</OntologyResource>
05     <OntologyNameSpace>http://...classes.owl#</OntologyNameSpace>
06   </Operation>
07   <Inputs msg='ReqMsg'>
08     <Input id='0'>
09       <Name>name</Name>
10       <OntologyResource>ADLCityName</OntologyResource>
11       <OntologyNameSpace>http://...classes.owl#</OntologyNameSpace>
12     </Input>
13   </Inputs>
14   <Outputs msg='ResMsg'>
15     <Output id='0'>
16       <Name>output</Name>
17       <OntologyResource>ADLPoint</OntologyResource>
18       <OntologyNameSpace>http://...classes.owl#</OntologyNameSpace>
19     </Output>
20   </Outputs>
21 </FunctionalitySection>
```

Figura 4.4: Aspectos funcionales del componente integrado `CI-Gazetteer` en formato ICML

Mucho más interesante resulta la declaración de los aspectos funcionales. El usuario describe la signatura que ofrece el nuevo componente integrado, es decir, el nombre de la operación y los parámetros de entrada y salida. La Figura 4.4 muestra el extracto de código ICML resultante tras definir los atributos funcionales del componente integrado `CI-Gazetteer` a partir del servicio web `ADL Gazetteer`. La declaración de la operación (líneas 02-06) incluye el nombre sintáctico de la operación (línea 03) y su anotación semántica (líneas 04-05), es decir, el concepto que define la operación `getCoordinates`. La funcionalidad de esta operación es anotada mediante el concepto `LocSpat` (línea 04), que define una operación que devuelve un atributo de tipo espacial basado en una localización. Este concepto forma parte de una ontología de dominio [85] accesible

a través de su namespace (línea 05). De forma similar, el usuario completa la definición de la signatura especificando los parámetros de entrada (líneas 07-13) y de salida (líneas 14-20) de la operación `getCoordinates`. El parámetro de entrada `name` (línea 09) queda asociado al concepto `ADLCityName` (línea 10) mientras que el parámetro de salida `output` (línea 16) al concepto `ADLPoint` (línea 17), ambos conceptos pertenecen a la misma geo-ontología que hemos utilizado previamente. Es interesante destacar que la signatura de la operación no está explícitamente enlazada con ninguno de los tipos de datos simples en XML (*integer*, *string*, etc.) o tipos compuestos definidos en descripciones WSDL, como ocurre en la declaración de los servicios web o en los lenguajes de composición de servicios web. La descripción ICML únicamente proporciona una relación a nivel semántico, anotando la signatura del nuevo componente integrado con conceptos de la ontología de dominio.

En el capítulo 3 comentábamos algunos criterios que afectan al grado de reutilización de un componente integrado, como el nivel de granularidad y el de dependencia de datos. Justo en esta fase de creación, es importante tener en cuenta estos criterios para que la operación del nuevo componente integrado se ajuste convenientemente al nivel de granularidad de las operaciones de los servicios web. Definíamos granularidad como el grado de refinamiento funcional de un componente integrado. Es probable que varias operaciones de diferentes servicios web pertenezcan al mismo componente integrado. Por ejemplo, varios servicios web se agruparán bajo el mismo componente integrado `CI-Gazetteer`, proporcionando la operación genérica `getCoordinates` que en cada servicio web puede aparecer con nombre distinto sintácticamente como `getLocation` o `getCoordinatesXY`. También, dada una operación *coarse-grained* de un servicio web, partes del resultado de esta operación pueden pertenecer a diferentes componentes integrados. Con la descomposición funcional de una operación compleja (*coarse-grained*) en partes más sencillas (*fine-grained*), favorecemos la flexibilidad porque añadimos más alternativas posibles a un componente integrado [96].

Con la especificación de los atributos descriptivos y funcionales hemos definido la interfaz pública del componente integrado. Aunque todavía queda por definir la interfaz privada –sus atributos estructurales y de enlace–, ya es posible derivar su descripción funcional en formato WSDL-S. Desde un punto de vista práctico, este nuevo componente integrado ya puede ser reutilizado en otras composiciones en el momento en que se registra su descripción funcional en el repositorio de servicios web. Aquí se aprecia la importancia de la encapsulación en los componentes integrados, separando en dos interfaces independientes la funcionalidad que ofrece de cómo está estructurada internamente.

Los patrones de selección, incluidos en los aspectos estructurales de un componente integrado, aportan cierto grado de flexibilidad a un componente integrado. Cabe recordar que dividíamos los patrones, analizados en el capítulo anterior, en patrones de *selección* y de *composición*. Los patrones de selección son utilizados en el proceso de abstracción mientras que los patrones de com-

posición son válidos durante el proceso de composición. El usuario selecciona uno de los dos posibles patrones de selección, bien el patrón AND-DISC o OR-DISC, asegurando así que tan solo uno de los resultados obtenidos del conjunto de servicios web candidatos será considerado en tiempo de ejecución.

El último paso en la creación de un componente integrado consiste en la especificación de los aspectos de enlaces. El usuario establece las transformaciones de datos y filtros necesarios entre los parámetros de cada servicio web con los parámetros del nuevo componente integrado. La idea consiste en establecer, mediante expresiones XPath, la relación entre la operación que acabamos de definir para el componente integrado con las operaciones de los servicios web. Evidentemente, cuanto mayor sea la diferencia de granularidad entre ambas operaciones, mayor dificultad presentará la definición de los enlaces entre ellas. Por eso, es importante establecer un nivel óptimo de granularidad entre la operación (abstracta) del componente integrado y las diferentes operaciones (concretas) de los servicios web.

Finalmente, durante el proceso de abstracción son válidos los enlaces de tipo *input*, *output*, y *constant*. Si se utilizaran enlaces *internal* durante el proceso de abstracción, éstos definirían el flujo de datos interno entre los propios servicios web que forman el nuevo componente integrado. Esto no tiene sentido durante el proceso de abstracción porque los servicios web deben ser independientes entre sí para ofrecer la máxima flexibilidad al componente integrado del que forman parte. El objetivo del proceso de abstracción no consiste en componer servicios web para que interactúen entre ellos, sino en agrupar servicios web según la funcionalidad que ofrecen para aumentar las probabilidades de éxito de la ejecución del componente integrado.

Creación automática

La creación automática genera un componente integrado automatizando la especificación de todos sus atributos a partir de un servicio web descrito mediante WSDL-S. Como el servicio web ya se encuentra anotado semánticamente, es posible automatizar tanto la anotación de la operación y los parámetros del nuevo componente integrado como la declaración de los enlaces entre los parámetros. Sin embargo, este mecanismo es más estricto que la creación manual porque únicamente permite la creación de un componente integrado a partir de un único servicio web anotado. Puede imaginarse que la creación automática facilita la creación de un componente integrado "plantilla", para que posteriormente el usuario pueda asociar manualmente más servicios web al nuevo componente integrado.

La creación automática requiere la especificación de ciertos valores por defecto para los atributos que definen un componente integrado. Por ejemplo, la Figura 4.5 muestra la creación automática del componente integrado `CI-Gazetteer` a partir del servicio web `ADL Gazetteer`. Tanto la operación `getCoordinates` como el parámetro de entrada `name` y de salida `output` del

Figura 4.5: Creación automática de un componente integrado durante el proceso de abstracción

servicio web tienen ya asociado un concepto. En la creación automática, el nombre de la operación del componente integrado y el de sus parámetros es idéntico al del servicio web inicial. Así, la operación `getCoordinates` (recuadro rojo), y los parámetros `name` (recuadro verde) y `output` (recuerdo azul) son copiados literalmente en la descripción WSDL-S del componente integrado. Sin embargo, esta vez el servicio web `ADL Gazetteer` tiene un segundo parámetro de entrada `payment` que indica la forma de pago. A diferente de los otros parámetros, el parámetro `payment` no está anotado, es decir, no tiene un concepto asociado. Por este motivo, este parámetro no aparece en la operación del componente integrado `CI-Gazetteer`. Por lo tanto, los parámetros de entrada o salida del servicio web que estén anotados pasarán automáticamente al componente integrado. En cambio, los parámetros sin anotación semántica no aparecerán en el componente integrado que se genera.

Otra alternativa consiste en añadir al componente integrado los parámetros no anotados pero asociándolos el concepto raíz `Thing`, ya que todo concepto de una ontología se deriva de `Thing` (al menos en OWL). En ese caso, `payment` formaría parte del componente integrado `CI-Gazetteer` de la Figura 4.5. Sin embargo, hemos considerado la primera opción porque el concepto `Thing` no aporta nada nuevo en la definición de un parámetro. Además, el hecho que el parámetro `payment` no esté incluido en la operación del componente integrado

no significa que no haya que tenerlo en cuenta, ya que este parámetro deberá estar presente cuando el servicio web `ADL Gazetteer` sea invocado. Será necesario entonces especificar de algún modo un valor por defecto para los parámetros no anotados de los servicios web. Para resolver este problema, generamos enlaces *constant* que asignan un valor constante a un parámetro (por ejemplo `payment=‘‘cash’’`), que será utilizado durante el proceso de transformación (véase sección 4.4).

La Figura 4.5 muestra también como se generan los restantes atributos que definen un componente integrado. El patrón AND-DISC sirve de patrón por defecto para la definición de los atributos estructurales. Los enlaces de tipo *input* y *output* entre parámetros se establecen de forma automática en función del concepto asociado. Parámetros de entrada o de salida con el mismo concepto se enlazan automáticamente, generándose un enlace *input* entre los parámetros de entrada del servicio web `ADL Gazetteer` y del componente integrado `CI-Gazetteer` porque ambos tiene asociado el concepto `ADLCityName`. Del mismo modo, se genera un enlace *output* porque los parámetros de salidas están etiquetados con el mismo concepto `ADLPoint`.

4.2.3. Registro de componentes integrados

Para que un componente integrado pueda ser reutilizado debe ser registrado en un repositorio. Es irrelevante para nuestra aproximación si es registrado en un repositorio público o privado. Esto dependerá de cada caso en particular. Supongamos un diseñador en una empresa que, por razones de seguridad, crea ciertos componentes integrados con funcionalidad crítica y los registra en un repositorio privado para su uso interno (Intranet). Tal vez, otros componentes integrados de propósito general son registrados en repositorios públicos, accesibles para el resto de usuarios externos (por ejemplo clientes). La utilización de un repositorio público o privado afecta a la capacidad de reutilizar componentes integrados desde el punto de vista del usuario final. El usuario externo solo puede utilizar componentes integrados del repositorio público, no del privado. Sin embargo, desde el punto de vista teórico planteado en esta tesis, la capacidad de reutilización únicamente depende del hecho de registrar un componente integrado en un repositorio para que puede ser localizado, independientemente de que su acceso esté o no limitado por razones ajenas.

La Figura 4.6 muestra los repositorios donde se registran las descripciones generadas para un componente integrado. Una vez especificada completamente la interfaz pública en la fase de creación, derivamos la descripción WSDL-S que captura la semántica introducida en la signatura del componente integrado (atributos funcionales). La descripción WSDL-S se registra en un repositorio de servicios web que contiene tanto descripciones WSDL como WSDL-S, ya que ambas son compatibles [138]. De este modo, tanto los servicios web como los componentes integrados que se vayan creando pueden ser localizados en los mismos repositorios mediante los mismos mecanismos de consulta.

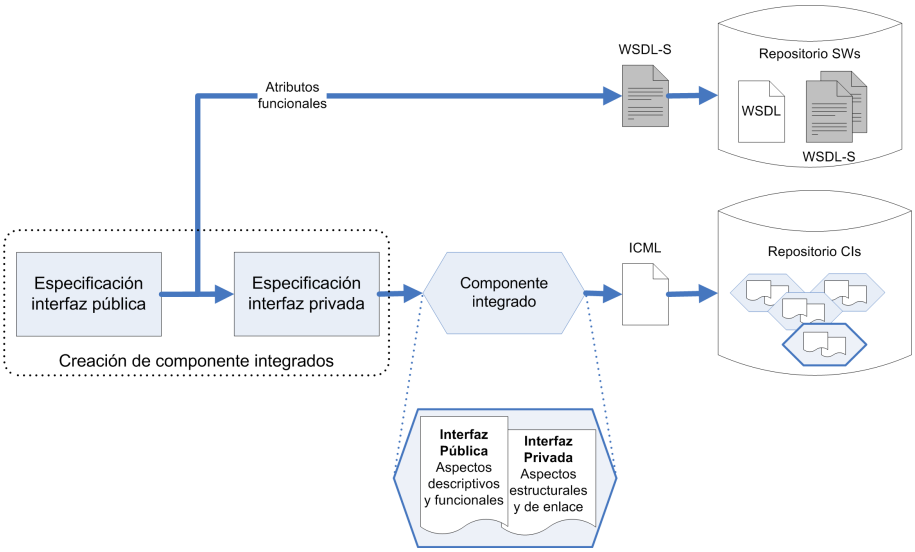


Figura 4.6: Registro de un componente integrado durante el proceso de abstracción

Una vez completada la creación de un componente integrado, registramos la descripción ICML en un repositorio de componentes integrados. La descripción ICML es una representación interna que almacena todos los atributos de un componente integrado (descriptivos, funcionales, estructurales y de enlaces) y cuyo esquema formal se encuentra en el Anexo A. Es importante indicar que la descripción ICML no es visible para el usuario final, el cual únicamente conoce la descripción WSDL-S para localizar y reutilizar componentes integrados. Evidentemente, será necesario acceder a la descripción ICML si deseamos realizar cambios en los atributos de un componente integrado. Sin embargo la edición de una descripción ICML es transparente al usuario porque se realiza mediante una herramienta software descrita en el capítulo 5. Por lo tanto, un componente integrado queda representado por dos descripciones: la descripción WSDL-S para la interfaz pública y registrada en repositorios de servicios web accesibles para los usuarios; y la descripción ICML que es una representación completa e interna (no visible) del componente integrado.

4.3. Proceso de composición

El proceso de composición permite a un usuario construir nuevos componentes integrados con mayor funcionalidad (composiciones) reutilizando componentes integrados ya existentes. El proceso de composición debe entenderse como un proceso *incremental*, donde en cada iteración nos vamos acercando a

la solución de nuestro problema, es decir, vamos construyendo gradualmente componentes integrados hasta alcanzar la funcionalidad requerida.

Una posible alternativa al proceso de composición incremental consistiría en resolver el problema en un solo paso, combinando en una única composición todos los componentes integrados necesarios para resolver el problema. A pesar de que realizar una composición en un solo paso resulta un proceso de mayor complejidad desde el punto de vista del diseñador, también va generalmente en detrimento de la reutilización de los componentes integrados. La creación de composiciones grandes, con muchos componentes involucrados en un solo paso, impide que las partes internas de estas composiciones (subcomposiciones) sean reutilizadas por otros usuarios. Si procedemos de un modo gradual, creando composiciones que añadan mayor funcionalidad paso a paso, generamos componentes integrados con diferentes niveles de abstracción que quedan registrados en el repositorio para poder ser reutilizados con posterioridad.

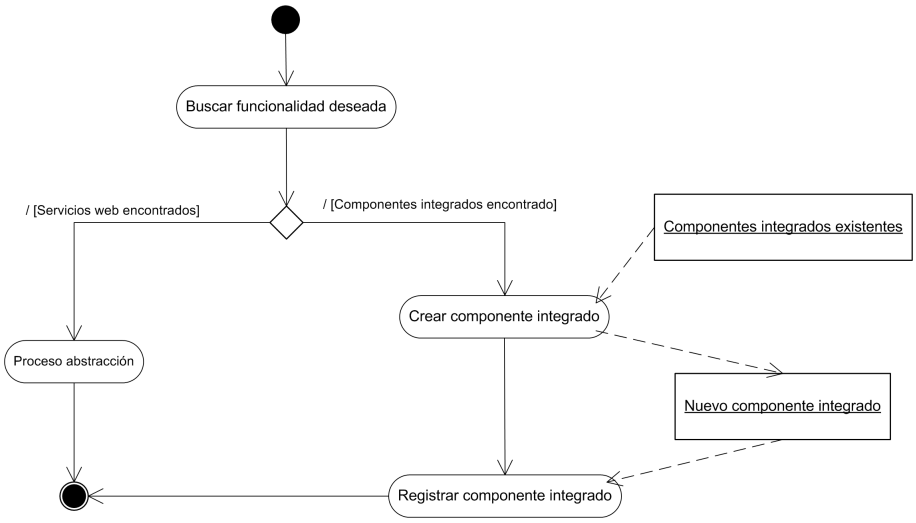


Figura 4.7: Proceso de composición

La Figura 4.7 muestra un diagrama de actividad UML para el proceso de composición. El usuario realiza también una búsqueda en el repositorio de servicios web en busca de servicios o componentes integrados que cumplan sus necesidades. Si únicamente se descubren servicios web, se debe iniciar el proceso de abstracción para crear previamente los correspondientes componentes integrados. Si la búsqueda devuelve componentes integrados, entonces iniciamos el proceso de composición. El escenario más habitual es que el usuario obtenga, como resultado de su búsqueda, una serie de componentes integrados que cubran parcialmente sus requisitos. Ninguno cubre los requisitos iniciales completamente, pero todos aportan algo. La forma de proceder es combinar

estos componentes integrados para formar nuevos componentes con mayor funcionalidad hasta acercarnos a los requisitos buscados.

Existen muchas semejanzas entre el proceso de abstracción y el proceso de composición. De hecho, las fases de selección, creación y registro son comunes en ambos procesos. Por ejemplo, la tarea de seleccionar bien servicios web (proceso de abstracción) o componentes integrados (proceso de composición) es en esencia idéntica en ambos procesos. También, la creación de un nuevo componente integrado es el objetivo de ambos procesos que finalmente queda registrado en el repositorio. Sin embargo, la separación entre ambos procesos no es simplemente por hacer el texto más legible, sino que cada proceso mantiene ciertas características de contexto que lo hacen singular. Aunque ambos procesos producen el mismo resultado (componentes integrados), los servicios web y los patrones de selección son utilizados en el proceso de abstracción, mientras que los componentes integrados creados previamente y los patrones de composición son considerados para el proceso de composición. Además, los componentes integrados generados por ambos procesos tienen diferentes niveles de abstracción. Un componente integrado resultado del proceso de composición no sabe nada de servicios web concretos porque no se encuentran en niveles adyacentes. En cambio, uno generado durante el proceso de abstracción queda definido por servicios web porque se encuentran en niveles contiguos. Por lo tanto, ambos procesos se basan en los mismos mecanismos aunque aplicados en niveles diferentes de la arquitectura (véase Figura 4.1). El simple hecho de unificar los mecanismos en ambos procesos simplifica considerable la creación de componentes integrados, simplificando al mismo tiempo la metodología de composición.

4.3.1. Selección de componentes integrados

A diferencia de la fase de selección durante el proceso de abstracción (sección 4.2.1), el usuario localiza componentes integrados como punto de partida para el proceso de composición. Sin embargo, tanto los servicios web como los componentes integrados son localizados inspeccionando sus descripciones funcionales –bien WSDL o WSDL-S respectivamente– ya que residen en el mismo tipo de repositorios. En definitiva, las etapas de selección son idénticas en ambos procesos excepto por el hecho que en el proceso de abstracción se recuperan servicios web mientras que componentes integrados en el proceso de composición.

4.3.2. Creación de componentes integrados

La creación de un componente integrado es la fase principal del proceso de composición y consiste también en la especificación de todos los aspectos que definen a un componente integrado. La Figura 4.8 muestra en detalle los diferentes actores involucrados en la creación de un componente integrado. A

diferencia de la fase de creación en el proceso de abstracción, aquí se inicia la creación de componentes integrados a partir de las descripciones funcionales de los componentes integrados (en formato WSDL-S) pero utilizando igualmente las mismas ontologías de dominio. Tampoco se distingue entre creación manual o automática, sino que únicamente planteamos la creación manual que denominamos simplemente como composición de componentes integrados. La progresiva aplicación del proceso de composición produce componentes integrados con diferentes niveles de abstracción y con funcionalidades heterogéneas.

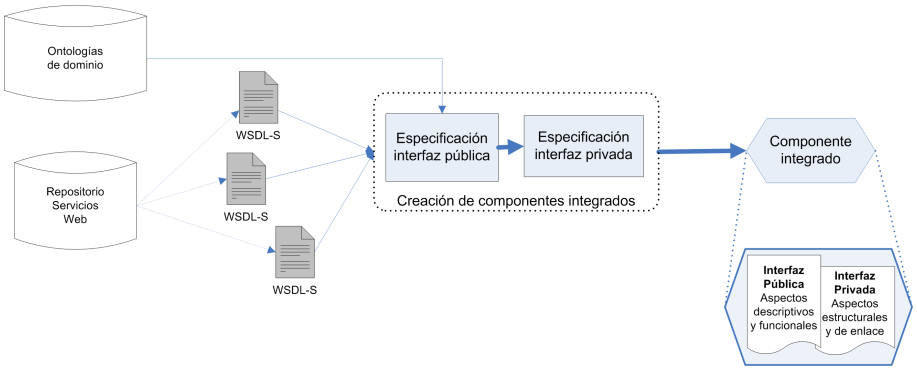


Figura 4.8: Creación de un componente integrado durante el proceso de composición

La Figura 4.8 muestra como la definición de un componente integrado se divide en la especificación de la interfaz pública y privada. La definición de la interfaz pública se realiza del mismo modo que en el proceso de abstracción, describiendo la signatura que ofrece el nuevo componente integrado en términos sintácticos y semánticos a partir de los componentes integrados del nivel inferior.

La interfaz privada engloba los aspectos estructurales y de enlace. Para la declaración de los aspectos estructurales se utiliza en esta ocasión los patrones de composición definidos en el capítulo 3. En el proceso de abstracción vimos la faceta menos conocida de los patrones mediante los patrones de selección porque éstos aportaban cierto grado de flexibilidad al componente integrado. Esta vez los patrones de composición muestran el lado más conocido: secuencias, paralelismo, bucles o condicionales. El usuario tan solo selecciona uno de los posibles patrones de composición (SEQ, AND, OR, XOR, LOOP-COND, LOOP-ITER) en la definición del componente integrado. Por ejemplo, utilizamos el patrón de composición SEQ para modelar los componentes integrados involucrados en secuencias, o AND para componentes integrados concurrentes, es decir, cada uno de ellos en ramas paralelas.

Lógicamente, la semántica del patrón de composición seleccionado establece los tipos de enlaces válidos. Por ejemplo, el patrón SEQ permite especificar los

enlaces *input*, *output* e *internal*. Los enlaces *internal* definen el flujo de datos entre componentes integrados en el mismo nivel, típico en una secuencia: la salida del primero será la entrada del segundo y así sucesivamente. Sin embargo, no existen enlaces *internal* cuando se utiliza el patrón AND porque cada rama de la bifurcación funciona de forma independiente (no existen flujos de datos entre componentes integrados al mismo nivel). En definitiva, mientras en el proceso de abstracción son posibles los enlaces *input*, *output* y *constant*, durante el proceso de composición pueden darse los enlaces de tipo *input*, *output* e *internal*.

4.3.3. Registro de componentes integrados

El registro de un nuevo componente integrado durante el proceso de composición es completamente idéntico al registro durante el proceso de abstracción explicado en la sección 4.2.3. Esto no resulta extraño puesto que el componente integrado es el único tipo de componente definido en nuestra aproximación. Parece lógico pues que un componente integrado quede registrado en los diferentes repositorios del mismo modo y genere las mismas descripciones (véase Figura 4.6), independientemente del proceso en el cual haya sido creado.

4.4. Proceso de transformación

Una vez creado el componente integrado requerido, bien aplicando únicamente el proceso de abstracción o bien realizando procesos de composición adicionales, el proceso de transformación permite convertir la descripción del componente integrado seleccionado en un proceso ejecutable WSBPEL [105] que, a su vez, podrá ser ejecutado por motores de ejecución compatibles. En principio, preferimos WSBPEL como lenguaje de salida porque puede ser considerado como el estándar del mundo empresarial para arquitecturas SOA [119]. Además, existen bastantes motores de ejecución disponibles para procesos WSBPEL como por ejemplo Oracle BPEL Process Manager[1] o los listados en `http://www.activebpel.org`.

La Figura 4.9 muestra un diagrama de actividad UML para el proceso de transformación. El primer paso consiste en localizar en el repositorio de servicios web el componente integrado que va a ser transformado que denominaremos componente integrado *raíz*. Si únicamente se descubren servicios web, se debe iniciar el proceso de abstracción para crear previamente los correspondientes componentes integrados. En caso contrario iniciamos el proceso de transformación. Tal como explicábamos en la introducción de este capítulo, debido a la relación de descomposición entre niveles adyacentes, el componente integrado raíz genera una estructura jerárquica en árbol en la cual un nodo padre queda definido completamente por sus nodos hijo del nivel inmediatamente inferior.

[1] `http://www.oracle.com/bpel`

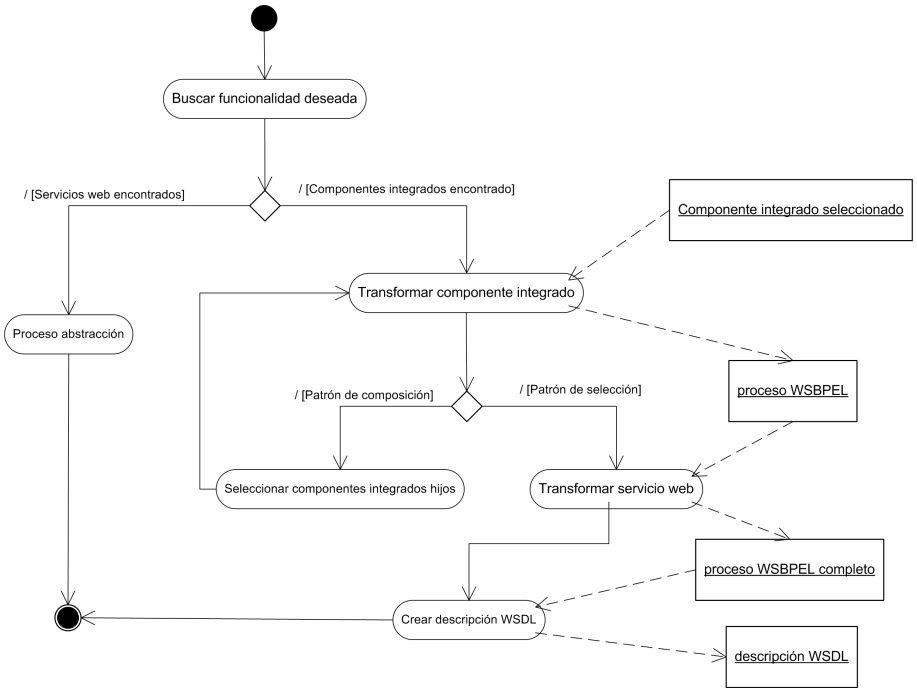


Figura 4.9: Proceso de transformación

El algoritmo de transformación recorre dicha estructura en árbol en busca de las representaciones de los nodos hijo, bien componentes integrados o servicios web. La condición de parada de la búsqueda será encontrar un componente integrado (nodo interno) con un patrón de selección, ya que esto indica que los nodos hijo de ese componente integrado serán servicios web, y por lo tanto nodos hoja. Para cada nodo visitado del árbol, se recupera la descripción del componente integrado y se transforma en código WSBPEL en función del patrón que incluye, como veremos en los siguientes apartados de esta sección. Una vez ensamblado completamente el proceso WSBPEL, cuando el árbol se ha recorrido completamente, se genera la descripción WSDL asociada al proceso WSBPEL resultante puesto que los motores de ejecución de WSBPEL requieren de ambas descripciones para la invocación del proceso.

4.4.1. Transformaciones generales

Antes de pasar a describir como los patrones de composición y de selección son transformados en su descripción WSBPEL equivalente, especificamos algunas transformaciones generales entre la estructura de un componente integrado y los constructores del lenguaje WSBPEL.

En general, el componente integrado raíz –la composición que va a ser transformada– se identifica como un proceso WSBPEL (`process`) que contiene ciertas propiedades por defecto. Por ejemplo, este proceso incluye una actividad `receive` seguida bien de una actividad `reply` para un proceso síncrono o bien de una actividad `invoke` para realizar la llamada asíncrona de *callback* al cliente (quien ejecuta el proceso) para procesos asíncronos. El proceso WSBPEL obtenido de este modo simplemente realiza una función *echo*, es decir, vuelca como salida la entrada del proceso sin realizar ningún procesamiento.

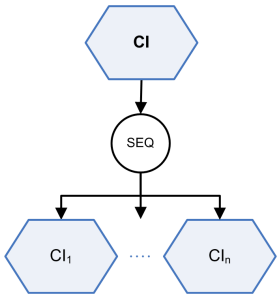
El conjunto de componentes integrados contenidos en el componente integrado raíz que incluyen un patrón de selección, junto con el usuario o cliente del proceso, definen los `partner links` que representan los diferentes actores que interaccionan con el proceso WSBPEL. Los parámetros de entrada y salida definidos en los aspectos funcionales son transformados como variables globales y propiedades del proceso WSBPEL. Los aspectos estructurales (patrones) son transformados en código WSBPEL como una combinación de actividades básicas y estructuradas así como de actividades `scope` y `compensation` cuando sea necesario. El flujo de datos definido por los aspectos de enlace se representa como transferencias de datos entre variables globales del proceso mediante actividades `assign`. Finalmente, una invocación de un servicio web se corresponde con una actividad `invoke` para comunicaciones síncronas o una combinación de actividades `invoke` y `receive` para el modo asíncrono. Una vez transformado completamente el componente integrado raíz en un proceso WSBPEL, debemos definir como último paso los puertos de comunicación (`portType` y `serviceLinkType`) del proceso generando su correspondiente descripción WSDL para que pueda ser interpretado por los motores de ejecución.

4.4.2. Transformación de los patrones de composición

Esta sección inspecciona en detalle como los patrones de composición propuestos en la sección 3.5.4 del capítulo anterior son transformados en lenguaje WSBPEL. Hay que tener en cuenta que los componentes integrados con patrones de composición siempre incluyen otros componentes integrados (no servicios web) como nodos hijo. En consecuencia las actividades WSBPEL que permiten la invocación de servicios web no son necesarias durante la transformación de los patrones de composición. Por razones de legibilidad, hemos incluido una función auxiliar *`toWSBPEL()`* (que no pertenece al lenguaje WSBPEL), que nos permitirá recorrer recursivamente la estructura en árbol generada por el componente integrado raíz, transformando cada componente integrado visitado en su representación WSBPEL correspondiente. El código WSBPEL mostrado a lo largo de esta sección se corresponde con la versión actual (2.0) [105] que, aunque se encuentra todavía en documento de trabajo en tiempo de redacción de esta tesis, incluye algunos constructores nuevos bastante interesantes respecto a las versiones previas (1.0 y 1.1).

PC1 – SEQ

La Figura 4.10(a) muestra la estructura en árbol de un componente integrado `CI` que puede incluir n componentes integrados ($\texttt{CI}_1 \ldots \texttt{CI}_n$) modelados con el patrón de composición SEQ. La Figura 4.10(b) muestra la transformación del patrón de composición SEQ en código WSBPEL. Los componentes integrados $\texttt{CI}_1 \ldots \texttt{CI}_n$ son procesados en el mismo orden en que son añadidos durante la creación del componente integrado `CI`. En WSBPEL, toda actividad incluida en la actividad `sequence` (línea 03-12) se ejecuta secuencialmente. El bloque definido por las líneas 07-09 se repite para cada $\texttt{CI}_i$ con $i = 2, \ldots, n$.



(a) Representación en árbol

```
01 <?xml version='1.0' encoding='UTF-8'?>
02 <scope name='SEQ'>
03   <sequence>
04     <!-- Read input from parent node-->
05     <assign>...</assign>
06     toWSBPEL(CI1)
07     <!-- Data flow -->
08     <assign>...</assign>
09     toWSBPEL(CIn)>
10     <!-- Write output to parent node!-->
11     <assign>...</assign>
12   </sequence>
13 </scope>
```

(b) Representación WSBPEL

Figura 4.10: Transformación del patrón de composición SEQ

PC2 – AND

El patrón de composición AND ejecuta paralelamente cada componente integrado $\texttt{CI}_i$ en una rama independiente. Gráficamente, la estructura en árbol es idéntica a la mostrada en la Figura 4.10(a), cambiando únicamente el correspondiente patrón de composición. Tal como muestra la Figura 4.11, el patrón de composición AND se corresponde con la actividad `flow` (línea 04-15) en WSBPEL. Cada rama completa un determinado $\texttt{CI}_i$, que a su vez está envuelto en

una actividad `sequence` (líneas 05-09 y 10-14). En términos de WSBPEL, se ejecutan paralelamente n actividades en paralelo, donde cada actividad es en sí misma una secuencia. Como antes, la secuencia definida por las líneas 10-14 se repite para cada $\texttt{CI}_i$ con $i = 2, \ldots, n$.

La Figura 4.11 muestra una posible transformación en formato WSBPEL similar a las propuestas en [161, 150]. A diferencia de estos estudios, la sincronización para las distintas ramas en paralelo se realiza añadiendo una actividad `assign` (línea 17) que recoge todas las salidas producidas por cada invocación de los $\texttt{CI}_i$. La forma de recorrer la estructura en árbol en profundidad, visitando primero a los nodos hijo que al padre, hace que la condición de bifurcación (actividad `flow`) se asocie con los componentes integrados $\texttt{CI}_i$ (nodos hijo) mientras que la condición de sincronización (actividad `assign`) se lleve a cabo en el componente integrado `CI` (nodo padre).

```
01  <?xml version='1.0' encoding='UTF-8'?>
02  <scope name='AND'>
03    <sequence>
04      <flow>
05        <sequence>
06          <!-- Read input from parent node-->
07          <assign>...</assign>
08          toWSBPEL(CI1)
09        </sequence>
10        <sequence>
11          <!-- Read input from parent node-->
12          <assign>...</assign>
13          toWSBPEL(CIn)>
14        </sequence>
15      </flow>
16      <!-- JOIN: Write output to parent node!-->
17      <assign>...</assign>
18    </sequence>
19  </scope>
```

Figura 4.11: Transformación del patrón de composición AND

PC3 – XOR

En este caso una composición ofrece dos o más componentes integrados $\texttt{CI}_i$ alternativos pero, basado en una condición, únicamente uno de éstos es seleccionado. Al igual que con el patrón AND, la especificación WSBPEL soporta directamente el patrón de composición XOR [161, 150]. Los anteriores trabajos plantean una solución en WSBPEL 1.1, basada en una combinación de sentencias `case` y `otherwise` incluidas en una actividad `switch`, de forma similar a una instrucción SWITCH común en lenguajes de programación (C o Java). Sin embargo, debido a que la actividad `switch` ha sido reemplazada por la actividad `if` en WSBPEL 2.0, proponemos como solución el listado de la Figura 4.12.

La actividad `if` (líneas 04-21) es una solución adecuada para el patrón XOR siempre que se trate de actividades `if-elseif` anidadas (línea 04, 13 y 21), puesto que únicamente una rama debe ser activada. Cada componente integrado $\texttt{CI}_i$ está envuelto en una secuencia en el cuerpo de una actividad `if` (líneas 07-11) o `elseif` (líneas 15-19). El primer componente $\texttt{CI}_1$ se activa si cumple la condición `cond` del patrón XOR. Si no es así, comprobamos la misma condición para el segundo $\texttt{CI}_2$ y así sucesivamente hasta el $\texttt{CI}_n$, repitiendo la estructura `elseif` (líneas 13-20). En el momento en que un $\texttt{CI}_i$ cumpla la condición `cond`, se activa ese componente integrado y la traza de la ejecución pasa a la línea 23 donde, como anteriormente, una actividad `assign` captura la condición de sincronización.

```
01 <?xml version='1.0' encoding='UTF-8'?>
02 <scope name='XOR'>
03   <sequence>
04     <if>
05       <condition>cond</condition>
06       <then>
07         <sequence>
08           <!-- Read input from parent node-->
09           <assign>...</assign>
10           toWSBPEL(CI1)
11         </sequence>
12       </then>
13       <elseif>
14         <condition>cond</condition>
15         <sequence>
16           <!-- Read input from parent node-->
17           <assign>...</assign>
18           toWSBPEL(CIn)>
19         </sequence>
20       </elseif>
21     </if>
22     <!-- JOIN: Write output to parent node!-->
23     <assign>...</assign>
24   </sequence>
25 </scope>
```

Figura 4.12: Transformación del patrón de composición XOR

PC4 – OR

La especificación WSBPEL no soporta directamente el patrón OR [140, 161], y tampoco parece que existan planes para incluirlo en el borrador de la última versión [105], aunque algunos autores [150, 161] han propuesto soluciones parciales al patrón OR. Vasko et al. [150] proponen soluciones separadas para el patrón de bifurcación (*split*) y para el de sincronización (*join*) en vez de una única solución para tratar conjuntamente la pareja de patrones. Por otra parte, Wohed et al. [161] llegan a plantear una posible solución conjunta

mediante construcciones basadas en grafos (constructores alternativos de WS-BPEL heredados de sus antecesores). Sin embargo, la gran desventaja de esta solución es su complejidad ya que a medida que n crece, es decir el número de componentes integrados $\texttt{CI}_i$, también aumenta considerablemente tanto la complejidad como la descripción del código WSBPEL resultante. Por razones de simplicidad, hemos decidido considerar el patrón OR como XOR cuando el lenguaje de composición de servicios web utilizado sea WSBPEL.

PC5 – LOOP-COND

Como muestra la Figura 4.13(a) los dos patrones de bucles (PC5-6) iteran sobre un único componente integrado $\texttt{CI}_1$ incluido en una composición `CI`. Para el caso del patrón de composición LOOP-COND, dicho $\texttt{CI}_1$ se repetirá mientras cierta condición *cond* se cumpla. Por ejemplo, en el listado de la Figura 4.13(b) se repite la transformación del componente integrado $\texttt{CI}_1$, el cual puede incluir a su vez un patrón de composición LOOP-COND generando de esta forma bucles anidados. La actividad `scope` (líneas 06-14) que envuelve al componente integrado $\texttt{CI}_1$ asegura que exista un contexto independiente en cada iteración. Las actividades `assign` de la línea 12 establece, si fuera necesario, flujos de datos entre diferentes iteraciones. Por último, la transferencia de datos hacia el componente integrado `CI` de nivel superior se lleva a cabo mediante actividades `assign` (línea 17).

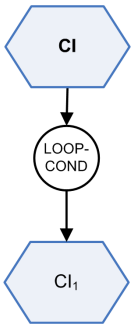
PC6 – LOOP-ITER

El patrón de composición LOOP-ITER es muy similar funcionalmente al patrón LOOP-COND, pero en este caso el componente integrado $\texttt{CI}_1$ se repite un cierto número n de veces. WSBPEL da soporte directo el patrón LOOP-ITER mediante la actividad `forEach`[2] (líneas 04-16) como muestra el listado de la Figura 4.14.

4.4.3. Transformación de los patrones de selección

Debido a que los patrones de selección interactúan con servicios web concretos, la transformación de estos patrones en su representación WSBPEL implica necesariamente añadir actividades básicas de invocación de servicios web, como `invoke` o `reply`, no tratadas en la transformación de patrones de composición. Además, como los servicios web representan los nodos hoja de la estructura en árbol que forma el componente integrado raíz, la función de transformación *`toWSBPEL()`* ya no es requerida porque no es posible seguir profundizando más en la estructura de árbol.

[2] Actividad nueva en WSBPEL 2.0



(a) Representación en árbol

```
01 <?xml version='1.0' encoding='UTF-8'?>
02 <scope name='LOOP-COND'>
03   <sequence>
04     <while>
05       <condition>cond</condition>
06       <scope>
07         <sequence>
08           <!-- Read input from parent node-->
09           <assign>...</assign>
10           toWSBPEL(CI1)
11           <!-- Data flow (feedback)-->
12           <assign>...</assign>
13         </sequence>
14       </scope>
15     </while>
16     <!-- Write output to parent node!-->
17     <assign>...</assign>
18   </sequence>
19 </scope>
```

(b) Representación WSBPEL

Figura 4.13: Transformación del patrón de composición LOOP-COND

PS1 – AND-DISC

Los actuales lenguajes de composición de servicios web no implementan el comportamiento del patrón *discriminator* original [140], incluyendo WS-BPEL, ni de forma directa ni con alguna solución alternativa como combinación de otros patrones existentes [150]. Sin embargo, como argumentábamos en la sección 3.5.3 del capítulo anterior, es posible proponer una solución parcial considerando la versión secuencial del patrón *discriminator* original. La última versión de WSBPEL permite una construcción especial que asocia a una actividad `invoke`, para la invocación de un servicio web, una actividad `compensationHandler` que actúa como una acción de compensación posterior a la invocación del servicio web. La acción de compensación será tenida en

```
01 <?xml version='1.0' encoding='UTF-8'?>
02 <scope name='LOOP-ITER'>
03   <sequence>
04     <forEach counterName='foo' parallel='no'>
05       <startCounterValue>1</startCounterValue>
06       <finalCounterValue>n</finalCounterValue>
07       <scope>
08         <sequence>
09           <!-- Read input from parent node-->
10           <assign>...</assign>
11           toWSBPEL(CI1)
12           <!-- Data flow-->
13           <assign>...</assign>
14         </sequence>
15       </scope>
16     </forEach>
17     <!-- Write output to parent node!-->
18     <assign>...</assign>
19   </sequence>
20 </scope>
```

Figura 4.14: Transformación del patrón de composición LOOP-ITER

cuenta si la actividad `invoke` se completa con éxito, es decir, si la invocación del servicio web no produce errores. Además WSBPEL también incorpora la gestión de errores que tiene lugar ante una invocación fallida de un servicio web. Por lo tanto, las acciones de compensación (`compensationHandler`) y la gestión de errores (`faultHandler`) son ortogonales en WSBPEL, es decir, cuando se completa la invocación de un servicio web a través del gestor de errores, dicha invocación nunca se considera una actividad completada satisfactoriamente, por lo que la acción de compensación asociada nunca tendrá lugar. Y al contrario, si una actividad de invocación se ejecuta con normalidad (sin errores), inmediatamente después la traza de ejecución pasa a la acción de compensación, sin tener en cuenta las acciones en el gestor de errores porque ningún error ha sido capturado.

La Figura 4.15(a) muestra la estructura jerárquica en árbol de un componente integrado `CI` con el patrón de selección AND-DISC, que a su vez tiene como hijos los servicios web que actúan de nodos hoja. El listado WSBPEL correspondiente a la transformación de patrón de selección AND-DISC se muestra en la Figura 4.15(b). En primer lugar se invoca el servicio web $\texttt{SW}_1$ (líneas 14-21). Si la invocación tiene éxito, la acción de compensación correspondiente (líneas 15-20) asigna el valor `false` a la variable local $\texttt{failService}_1$ (líneas 16-19). En este caso, el siguiente servicio web, $\texttt{SW}_2$, no será invocado porque la condición $\texttt{failService}_1$ (línea 23) será evaluada falsa. Si suponemos que la invocación del $\texttt{SW}_1$ produce un error, el gestor de errores (líneas 06-10) caza dicha excepción mediante la construcción `catchAll` (líneas 07-09), indicando que será capturado cualquier error independientemente del tipo que sea. La acción tomada ante cualquier error es la actividad `empty` (línea 08). Como su

nombre indica, la actividad `empty` no hace nada porque las variables locales que controlan el estado de la invocación para cada servicio web ($\texttt{failService}_i$) ya están inicialmente asignadas con el valor correcto ante un error, es decir, puesta a verdadero (líneas 04-05). Después de la gestión de errores, la traza de ejecución continúa con la actividad siguiente (`if`, líneas 22-38) a la actividad que produjo el error (`invoke`, líneas 14-21). Entonces, el siguiente servicio web $\texttt{SW}_2$ será invocado porque la condición $\texttt{failService}_1$ (línea 23) se evalúa cierta. El bloque `if` delimitado por las líneas 22-38 se repetirá para cada servicio web $\texttt{SW}_i$ con $i = 2, \ldots, n$.

Es importante subrayar que cada actividad de invocación está envuelta por estructuras condicionales `if` independientes pero no anidadas, como era el caso del patrón de composición XOR (véase Figura 4.12). Las estructuras condicionales anidadas (combinaciones `if` y `elseif`, nuevas en WSBPEL 2.0) no son idóneas en este caso porque la primera rama evaluada verdadera se toma sin inspeccionar las restantes alternativas, sin saber todavía si la invocación del servicio web seleccionado va a ser satisfactoria. Como ocurre con las instrucciones IF-ELSEIF en los lenguajes de programación tradicionales, las restantes ramas ya no son visitadas a pesar de que la rama seleccionada no pueda completarse con éxito.

Finalmente, las acciones de compensación asociadas a una invocación satisfactoria controlan también la transferencia de datos entre la salida que produce el servicio web invocado y el componente integrado `CI` que actúa de nodo padre (líneas 18 y 32). En definitiva, la representación WSBPEL de la Figura 4.15(b) nos permite modelar el patrón de selección AND-DISC como un discriminador secuencial que recoge los resultados del primer servicio web que finaliza con éxito, evitando la invocación de los restantes servicios web candidatos.

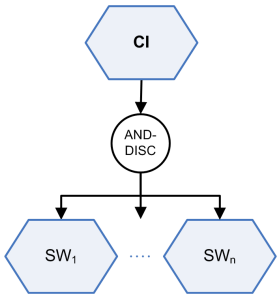
PS2 – OR-DISC

La solución propuesta para el patrón de selección OR-DISC es casi idéntica a la transformación presentada para el patrón AND-DISC (Figura 4.15(b)). El único cambio con respecto al anterior patrón es que ahora añadimos una condición adicional `cond`, basada por ejemplo en decisiones sobre datos del componente integrado, que filtra los servicios web que pueden ser invocados *a priori* de entre todos los candidatos. Por lo tanto, añadimos esta condición adicional al listado de la Figura 4.15(b), para asegurarnos que un servicio web será invocado si cumple ambas condiciones. Concretamente, debemos envolver la invocación del servicio web SW_1 en una estructura `if` para incluir explícitamente una condición (`<condition>`*cond*`</condition>`) y modificar la condición $\texttt{failService}_1$ (línea 23) del $\texttt{SW}_2$ para que quede como $\texttt{failService}_1$ `AND cond`.

4.5. Resumen

En este capítulo hemos presentado una arquitectura en niveles de abstracción para dar soporte al concepto de componente integrado y la metodología de composición. Hemos introducido el nivel de servicios abstractos entre el nivel de aplicaciones (procesos) y el nivel de servicios concretos (servicios web). Los componentes integrados residen en el nivel de servicios abstractos y mantienen una relación de descomposición entre ellos, de tal forma que un componente integrado de un nivel dado se *descompone* funcionalmente en los componentes integrados más simples del nivel inferior. Sin embargo, el concepto de componente integrado no dice nada acerca de cómo debe ser creado o compuesto. La metodología de composición complementa al componente integrado especificando los procesos básicos para el manejo de componentes integrados: el *proceso de abstracción* (o creación), el *proceso de composición* y el *proceso de transformación*.

Mediante el proceso de abstracción, creamos componentes integrados a partir de servicios web concretos, facilitando la flexibilidad puesto que agrupamos un conjunto de servicios web en un mismo componente integrado sin especificar cual de ellos será seleccionado en tiempo de diseño. Nuevos componentes integrados son generados durante el proceso de composición, reutilizando componentes integrado de niveles inferiores en la creación de nuevos componentes integrados de niveles superiores. Finalmente, mediante el proceso de transformación somos capaces de transformar la descripción de la estructura jerárquica que genera un componente integrado en un lenguaje de composición de servicios web como WSBPEL, listo para ser invocado mediante un motor de ejecución.



(a) Representación en árbol

```
01 <?xml version='1.0' encoding='UTF-8'?>
02 <scope name='AND-DISC' variableAccessSerializable='no'>
03   <sequence>
04     <!-- Init all local variables failServicei to true-->
05     <variables>...</variables>
06     <faultHandlers>
07       <cathAll>
08         <empty/>
09       </cathAll>
10     </faultHandlers>
11     <sequence>
12       <!-- Read input from parent node-->
13       <assign>...</assign>
14       <invoke name='SW1'>
15         <compensationHandler>
16           <assign>
17             <!-- Set false to failService1-->
18             <!-- Write SW1 output parent node-->
19           </assign>
20         </compensationHandler>
21       </invoke>
22       <if>
23         <condition>failService1</condition>
24         <then>
25           <sequence>
26             <!--Read input form parent>
27             <assign>...</assign>
28             <invoke name='SW2'>
29               <compensationHandler>
30                 <assign>
31                   <!-- Set false to failService2-->
32                   <!-- Write SW2 output parent node-->
33                 </assign>
34               </compensationHandler>
35             </invoke>
36           </sequence>
37         </then>
38       </if>
39     </sequence>
40   </sequence>
41 </scope>
```

(b) Representación WSBPEL

Figura 4.15: Transformación del patrón de selección AND-DISC

Capítulo 5

Implementación y resultados

En este capítulo presentamos una herramienta software para el diseño de componentes integrados. Dicha herramienta sigue la metodología de composición expuesta en el capítulo 4 para la creación, composición y transformación de componentes integrados.

La segunda parte de este capítulo incluye un estudio comparativo de los resultados obtenidos de nuestra aproximación atendiendo a criterios cualitativos como simplicidad, flexibilidad y reutilización comparado con WSBPEL, el estándar *de facto* de la industria para la composición de servicios web.

5.1. Introducción

La herramienta software, el Diseñador de Componentes Integrados (DCI), ha sido desarrollada para facilitar al usuario la creación, edición, composición, reutilización y transformación de componentes integrados. Para el diseño del DCI se han tenido en cuenta los siguiente criterios:

- *Facilidad de uso* e *intuitivo*. No hay duda de que existen muy buenas herramientas con interfaces gráficas para la composición de servicios web [53, 79, 121]. Sin embargo, un reto en el diseño del DCI ha consistido en que fuera intuitivo y fácil de manejar para un usuario inexperto. Creemos que, tras el diseño de unos pocos componentes integrados, un usuario es capaz de diseñar y reutilizar componentes integrados con funcionalidad avanzada. La unificación de los procesos de abstracción y composición, proporcionando en esencia el mismo procedimiento para la

creación y composición de componentes integrados, es un factor importante que permite al usuario asimilar rápidamente estos procedimientos, al mismo tiempo que disminuye la curva de aprendizaje del DCI.

- *Reutilización.* La reutilización es un aspecto esencial por dos razones. En primer lugar, la reutilización de componentes integrados es el objetivo principal para la definición del concepto de componente integrado (véase capítulo 3) y de la metodología de composición (véase capítulo 4). Esta característica ha sido trasladada al DCI haciendo casi indistinguible la utilización o reutilización de componentes integrados desde la perspectiva del usuario. Por otra parte, no solo se enfatiza la característica de reutilización de los componentes integrados, sino que la propia implementación ha sido desarrollada siguiente esta filosofía mediante el uso de *plug-ins* de Eclipse [36]. La plataforma Eclipse sigue un diseño modular integrando cientos de trozos de funcionalidad (*plug-ins*) de forma que puedan ser fácilmente reutilizados para construir aplicaciones. De esta forma, el DCI ha sido implementado tanto a partir de nuevos plug-ins como reutilizando otros ya existentes.

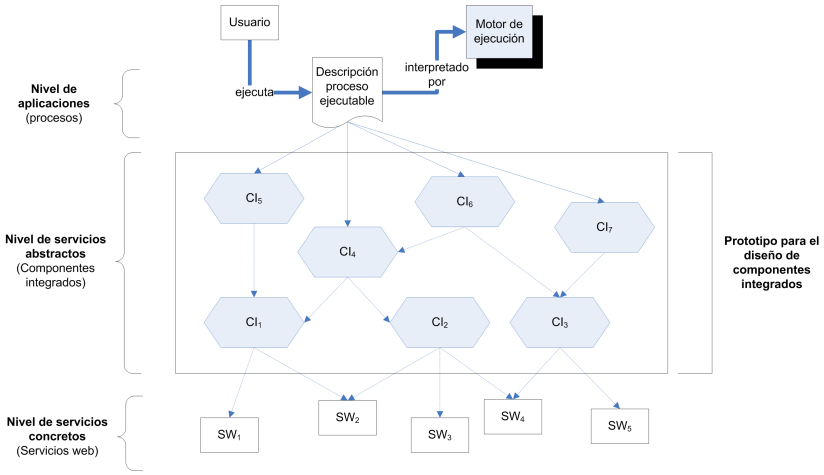


Figura 5.1: Relación arquitectura - DCI

La Figura 5.1 muestra la relación entre la arquitectura en niveles de abstracción descrita en el capítulo 4 y la herramienta DCI. Como se ha discutido en el capítulo anterior, hemos añadido un nivel de abstracción adicional entre el nivel de aplicaciones (procesos) y el de servicios concretos (servicios web) que se corresponde con el nivel de servicios abstractos o de componentes integrados. Justo en este nivel intermedio se sitúa el DCI para la creación, composición y transformación de componentes integrados.

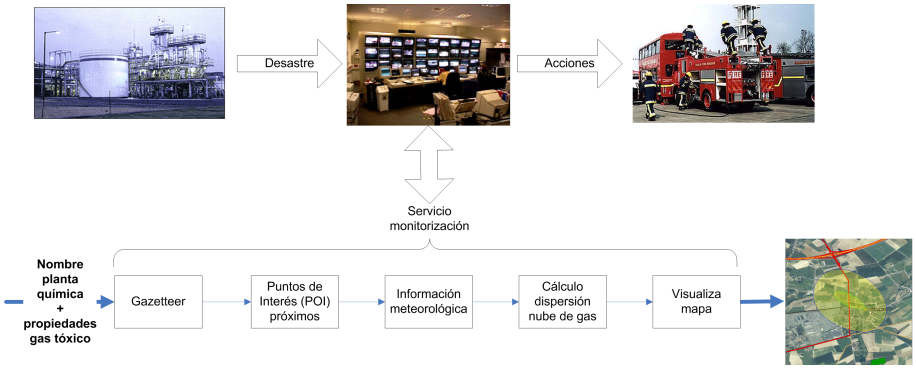
La primera parte de este capítulo describe la implementación del DCI, exponiendo su arquitectura y un ejemplo concreto de aplicación en un escenario de control de emergencias. La segunda parte se centra en el estudio comparativo llevado a cabo entre nuestra aproximación y WSBPEL.

5.2. Escenario: control de emergencias

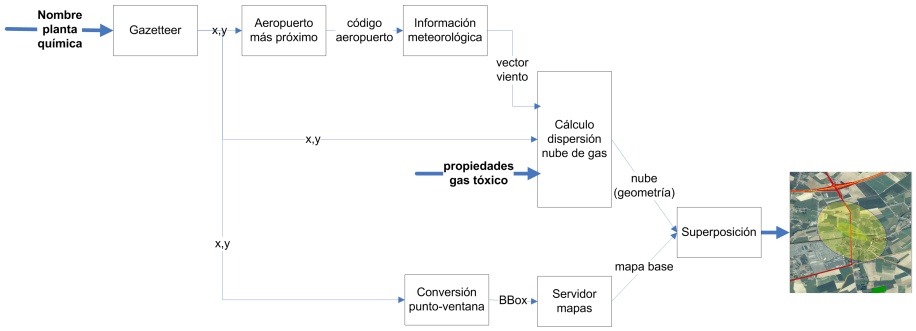
Cuando se produce un desastre o un evento inesperado, como por ejemplo el vertido del *Prestige* en las costas de Galicia o el huracán *Katrina* que arrasó recientemente Nueva Orleans, los centros de control de emergencias necesitan tener acceso inmediato a diferentes fuentes de datos que tienen relevancia sobre el área afectada. Estas fuentes de datos incluyen normalmente información y servicios geo-espaciales necesarios para determinar la magnitud del desastre, así como para facilitar tanto la coordinación de recursos humanos disponibles como la decisión sobre las acciones a llevar a cabo durante la respuesta inmediata al desastre [42].

Tradicionalmente los equipos de emergencia desplazados al lugar del desastre llevan consigo equipos portátiles con bases datos cartográficas estáticas, es decir, configuradas previamente con la información de la zona afectada. Sin embargo, la capacidad de acceder de forma *ad hoc* a diferentes servicios web y combinarlos sobre la marcha no solo añade un factor importante de flexibilidad sino que también permite el acceso a información geo-espacial actualizada, como información meteorológica o de control de tráfico, a medida que se encuentra disponible en los correspondientes servidores de datos. Existe una clara necesidad de mejorar las infraestructuras disponibles para que permitan, tanto a los analistas como a los demás profesionales de los servicios de emergencias (policía, bomberos, etc.), la combinación *ad hoc* de datos y servicios geo-espaciales durante situaciones de emergencia [13, 130].

Las Infraestructuras de Datos Espaciales (IDE), respaldadas por la propuesta INSPIRE del Parlamento Europeo para establecer una infraestructura espacial para todos los estados miembros de la Comunidad Europea, facilitan la disponibilidad y el acceso a los datos geo-espaciales a nivel local, regional, nacional y global [104]. Sin embargo, una IDE por sí sola resulta insuficiente si no existen aplicaciones que la exploten adecuadamente. Se necesitan mecanismos que permitan la localización, composición y reutilización de datos y servicios geo-espaciales para crear nuevos servicios compuestos y aplicaciones. Esto permite que los usuarios aprovechen con mayor eficiencia los recursos geo-espaciales, tanto datos como servicios, que proporciona una infraestructura de datos como puede ser una IDE. Nuestro caso de aplicación muestra como la reutilización de componentes integrados es un mecanismo viable para la creación de aplicaciones basadas en servicios encima de una IDE, reutilizando datos y servicios geo-espaciales proporcionados por este tipo de infraestructuras de datos.



(a) Actividades genéricas



(b) Flujo de datos entre actividades

Figura 5.2: Servicio compuesto de *monitorización*

El escenario propuesto tiene lugar en el contexto de situaciones de emergencia, desarrollado durante el proyecto europeo ACE-GIS [126]. Nos ponemos en el lugar de Abel, un técnico informático perteneciente a un centro de control de emergencias regional. La situación de emergencia comienza cuando se produce un accidente en un planta química cercana a una población liberando una nube de gas tóxica. Los analistas del centro de control de emergencias deciden que una de las tareas prioritarias debe consistir en monitorizar la nube producida por el escape de gas tóxico, para determinar en todo momento si la población local se encuentra en situación de riesgo y, en consecuencia, poner en marcha los correspondientes planes de evacuación. Abel, como responsable de la tarea de monitorizar la nube de gas tóxica, decide crear un servicio compuesto de *monitorización* para la nube de gas tóxica de forma que, a partir del nombre de la planta química y de las propiedades físicas del gas tóxico emitido, se obtenga una representación de la dispersión de la nube de gas tóxica sobre un mapa de la zona afectada. Realmente, la Figura 5.2(a) representa una idea aproximada de las actividades que formarán parte del servicio de *monitorización*. Sin embargo,

a medida que perfilamos más detenidamente ciertos detalles, como puede ser el flujo de datos entre las actividades, aumenta la complejidad del plan debido principalmente a que pueden aparecer nuevas actividades que anteriormente no resultaban necesarias

La Figura 5.2(b) representa de nuevo el plan para el servicio de *monitorización* pero con mayor nivel de detalle, mostrando esta vez el flujo de datos entre las diferentes actividades. La primera actividad consiste en un `gazetteer`, que toma como entrada un topónimo o un nombre de una institución (como por ejemplo el nombre de la planta química en cuestión), y devuelve una lista de posibles localizaciones geográficas de dicho topónimo. Típicos resultados son direcciones de calles o pares de coordenadas x,y expresadas en un sistema de referencia conocido como puede ser longitud/latitud. Por simplicidad, asumimos que la actividad `gazetteer` devuelve una única localización expresada como longitud/latitud para el topónimo buscado. Con las coordenadas x,y de la planta química en la mano, Abel requiere de una actividad que proporcione `información meteorológica`. Tanto la velocidad como la dirección del viento son parámetros necesarios para estimar la dispersión de la nube de gas. Sin embargo, Abel se da cuenta que no dispone de una estación meteorológica en la región. Para resolver este problema, decide utilizar previamente una actividad que le permita identificar los puntos de interés (POI) más cercanos a partir de unas coordenadas x,y conocidas, similar por ejemplo al servicio *Microsoft MapPoint Web Service*[1]. Tomando como puntos de interés los aeropuertos, ya que todos los aeropuertos disponen de estaciones meteorológicas, la actividad `aeropuerto más próximo` devuelve justamente la información del aeropuerto más cercano a las coordenadas de la planta química, como por ejemplo la localización, elevación o el código internacional de aeropuerto[2]. Ahora Abel ya dispone de la información correcta para interrogar la actividad `información meteorológica` del aeropuerto mediante su código y recuperar, entre otros datos meteorológicos, el valor concreto de la magnitud y dirección del viento.

El plan continúa con la actividad del `cálculo de la dispersión de la nube de gas`. Esta actividad requiere como entrada el vector del viento, la localización de la planta, y algunas propiedades físicas del gas (como la densidad) para calcular una estimación de la nube de gas en forma de un polígono. Cabe destacar en este punto que durante el desarrollo del proyecto ACE-GIS, uno de los socios (IONIC Software[3]) facilitó un servicio web para el cálculo aproximado de la dispersión de una nube de gas, interrogando internamente a un Web Feature Service (WFS) [112] según la especificación definida por el OGC, permitiendo filtrar y recuperar representaciones vectoriales de características geo-espaciales (*features*) como puntos, líneas o polígonos, como es el caso de este escenario.

[1] `http://www.microsoft.com/mappoint/default.mspx`
[2] Ejemplos de información asociada a aeropuertos puede encontrarse en `http://www.world-airport-codes.com/`
[3] `http://www.ionicsoft.com/`

Las anteriores actividades permiten calcular la dispersión de la nube de gas. Las siguientes dos actividades (parte inferior Figura 5.2(b)) se encargan de recuperar una imagen de la zona afectada que servirá de mapa base para superponer la nube de gas encima. Abel necesita un servidor de mapas similar a un Web Map Service (WMS) [113] que proporcione un mapa dada una ventana de visualización (*bounding box*). Aunque un WMS requiere diferentes parámetros de entrada para llevar a cabo una petición *GetMap*, por razones de simplicidad, consideraremos únicamente en este ejemplo el parámetro de entrada `bbox` (*bounding box*), asumiendo valores por defecto para los restantes parámetros. Como paso previo a la actividad `servidor mapas`, Abel utiliza una operación de geometría básica punto-polígono mediante la actividad `conversión punto-ventana` que permite generar automáticamente una ventana de visualización (polígono) a partir de unas coordenadas x,y (punto). Finalmente, la actividad de `superposión` superpone la nube de gas calculada (polígono) sobre el mapa base de la zona afecta (resultado de la actividad `servidor mapas`) produciendo un mapa de monitorización como el mostrado en la Figura 5.2(b).

Como comentario adicional, aunque no cubierto en este escenario, sería interesante que Abel pudiera realizar ciertas operaciones interactivas sobre el mapa generado por el servicio de *monitorización*. Por ejemplo, la visualización de las posibles rutas de evacuación (a partir de una consulta con un servidor de datos de carreteras) o la posibilidad de determinar la población en situación de riesgo (superponiendo por ejemplo otras capas del censo de la población). Como puede apreciarse, las posibles aplicaciones que pueden beneficiarse de una IDE son inimaginables y de gran valor añadido, tanto para usuarios individuales, empresas privadas como para las administraciones públicas [11].

5.3. DCI: diseñador de componentes integrados

En esta sección introducimos el DCI para el diseño de componentes integrados como un conjunto de vistas y editores gráficos. Primero, describimos los elementos de la arquitectura y su correspondencia en las diferentes vistas y editores del DCI. Luego nos concentramos en los aspectos más prácticos de la generación de componentes integrados para el escenario planteado anteriormente en la sección anterior.

5.3.1. Arquitectura

El DCI sigue el paradigma *Model-View-Controller* (MVC) [80]. Como comentamos anteriormente, el DCI está compuesto en realidad de un conjunto de plug-ins de Eclipse. Ya que la plataforma Eclipse sigue el paradigma MVC, parece lógico que nuestra decisión fuera también este paradigma para la implementación del DCI. Evidentemente existen otras alternativas de implementación perfectamente válidas (Visual Basic o Web) que seguramente derivarían en

una arquitectura del DCI completamente diferente. Sin embargo, creemos que la plataforma Eclipse es una buena elección puesto que se está convirtiendo en la plataforma dominante debido principalmente a su elegante arquitectura y a su capacidad para integrar diferentes lenguajes de programación, aplicaciones y plug-ins de terceras compañías [51].

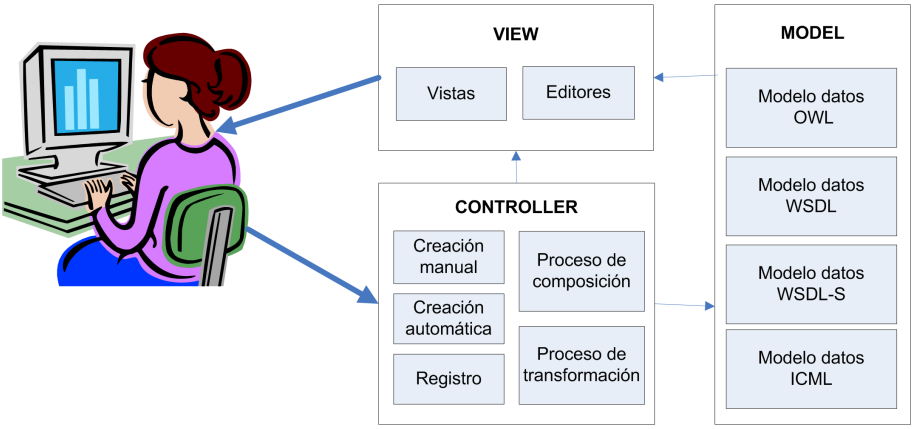


Figura 5.3: Arquitectura *Model-View-Controller*

El principio básico del paradigma MVC consiste en gestionar separadamente la entrada de datos, el modelo lógico de la aplicación y la respuesta visual debida a cambios en el estado del modelo. La Figura 5.3 muestra las capas básicas que integran el MVC: la capa *Model*, la capa *View* y la capa *Controller*. La capa *Model* contiene información de los componentes de la aplicación. La capa *View* es una representación visual de un componente, gestionando la interfaz de usuario. Digamos que la capa *View* es únicamente la parte visual de un componente. La capa *Controller* se encarga de todas las acciones, interpretando la entrada del usuario y actuando de nexo entre las capas *View* y *Model*. En la capa *Controller* tiene lugar la lógica de la aplicación, es decir, procedimientos y algoritmos para modificar el modelo en respuesta a eventos o para realizar cambios en el estado de un componente producidos en la capa *View*.

En la Figura 5.3 también aparecen los elementos que integran el DCI para cada una de las capas MVC. El DCI utiliza el Graphical Editing Framework (GEF[4]), un conjunto de plug-ins que agilizan el desarrollo de interfaces gráficas, para la implementación de las diferentes `vistas` y `editores` del DCI como parte de la capa *View*. Además, ya que GEF sigue el paradigma MVC como subproyecto del proyecto global de Eclipse, se acopla perfectamente con el diseño general propuesto. En la capa *Model* residen los diferentes modelos de datos para las ontologías (OWL), descripciones de servicios web (WSDL, WSDL-S) y las descripciones de componentes integrados (ICML). La definición de los

[4] `http://www.eclipse.org/gef/`

algoritmos y procedimientos para gestionar los componentes integrados queda definido en la capa *Controller*, como la `creación manual` o el `registro` de componentes integrados. Por ejemplo, cuando un usuario activa la acción de crear un nuevo componente integrado desde una vista de la capa *View*, los procesos de la capa *Controller* crearán una instancia de un componente integrado, la cual se almacenará en la capa *Model*, e informarán a la capa *View* que muestre el editor correspondiente para completar los atributos del nuevo componente integrado.

5.3.2. Vistas y editores

La siguiente sección da un breve repaso a los elementos gráficos que componen el DCI, relacionando los componentes de las capas del paradigma MVC (véase Figura 5.3) con su representación visual en el DCI. La funcionalidad de estos componentes será descrita con más detenimiento en la sección siguiente con la aplicación del DCI en el escenario de control de emergencias.

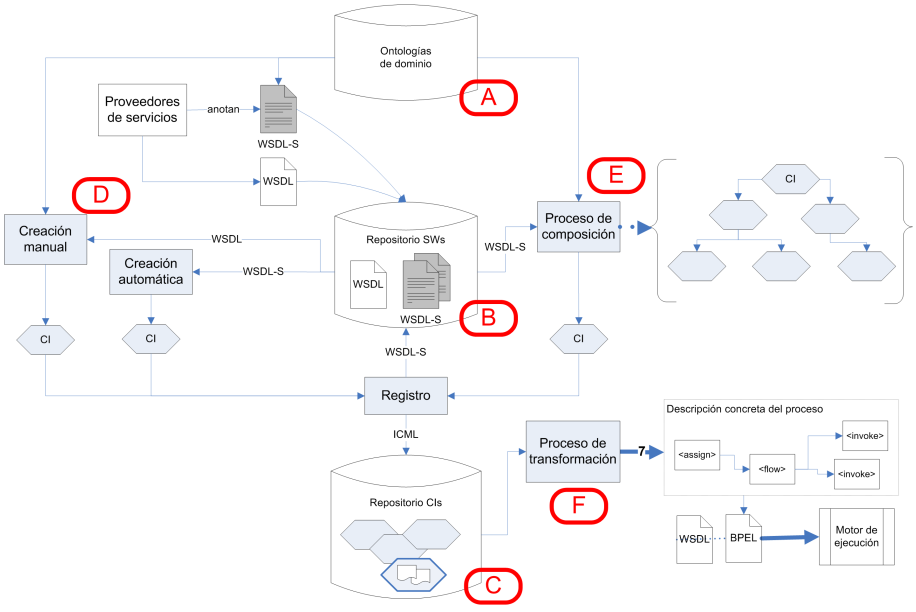


Figura 5.4: Interacciones en el DCI

La Figura 5.4 muestra como interaccionan los diferentes componentes del DCI. Lógicamente, estos componentes (plug-ins, algoritmos, procedimientos, etc.) se corresponden directamente con el concepto de componente integrado y la metodología de composición expuestos en los anteriores capítulos. Por ejemplo, la `creación manual` y la `creación automática` junto con el `registro` representan el proceso de abstracción (véase capítulo 4). La idea consiste en

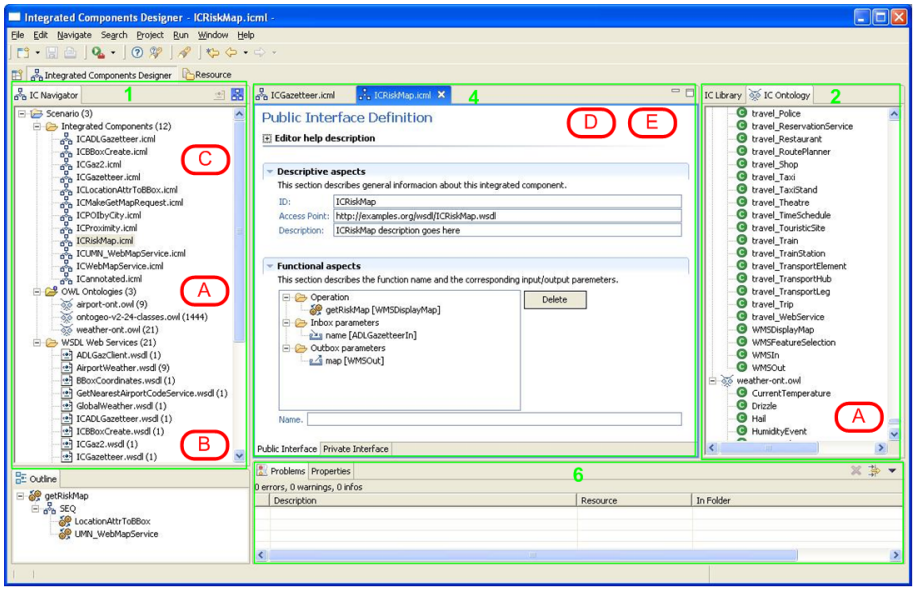


Figura 5.5: Vista DCI del editor para la interfaz pública

mostrar cómo son representados los componentes de la Figura 5.4 en las Figuras 5.5 y 5.6 del DCI. Para ello, hemos anotado estas tres figuras con diferentes letras y números. Las letras iguales establecen correspondencias directas entre componentes del paradigma MVC y su representación visual en el DCI. Los números únicamente identifican las diferentes vistas y editores que componen el DCI.

Los repositorios de la Figura 5.4 contienen los diferentes modelos de datos utilizados en la capa *Model*. El repositorio para las ontologías de dominio [A] contiene descripciones OWL, el repositorio de servicios web [B] almacena descripciones WSDL y WSDL-S, mientras que el de componentes integrados [C] contiene descripciones ICML. El contenido de estos repositorios es inspeccionado mediante la vista `IC Navigator` (1), en la parte izquierda de las Figuras 5.5 y 5.6. De esta forma, tal como muestra la Figura 5.5, la vista `IC Navigator` visualiza las ontologías [A], los servicios web [B] y los componentes integrados [C] de los diferentes repositorios en una única vista. Además, seleccionando una determinada ontología [A] en la vista `IC Navigator`, se muestran todos los conceptos (clases y propiedades) que contiene en la vista `IC Ontology` (2), situada en la parte derecha de la Figura 5.5. De forma similar, todas las operaciones [B] definidas por los servicios web y componentes integrados de la vista `IC Navigator` se listan en la vista `IC Library` (3) situada igualmente en la parte derecha de la Figura 5.6. El contenido de las vistas `IC Ontology`

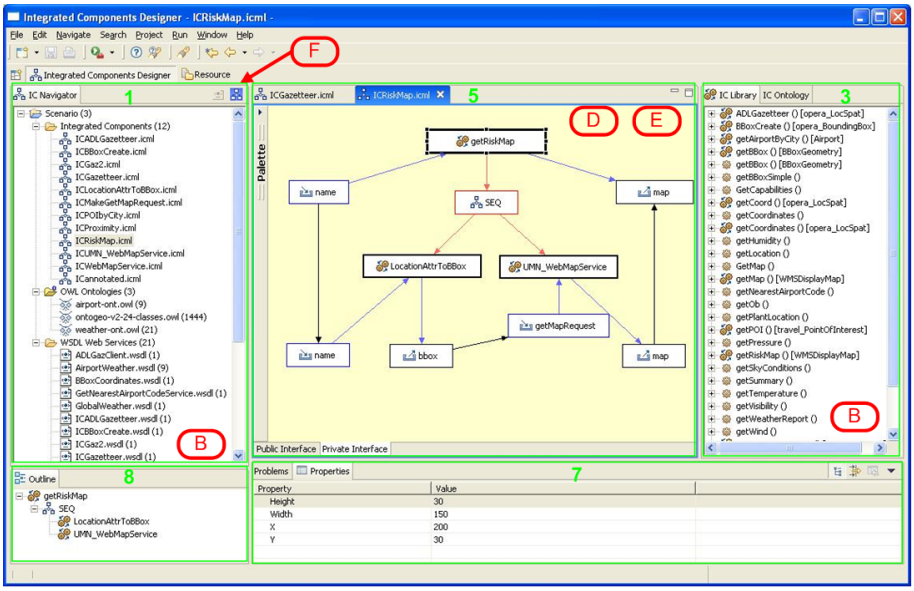


Figura 5.6: Vista DCI del editor para la interfaz privada

y `IC Library` está sincronizado con la vista principal `IC Navigator`. Cambios producidos en la última repercuten inmediatamente en las otras dos mediante el sistema de propagación de eventos de la capa *View* a la capa *Controller*.

Cuando un usuario decide crear un nuevo componente integrado mediante la creación manual [D] en la Figura 5.4, debe completar tanto la interfaz pública como privada de dicho componente integrado antes de registrarlo en el respectivo repositorio. El DCI proporciona dos editores gráficos [D] para la definición de un componente integrado: un editor para la interfaz pública (4 en Figura 5.5) y otro para la interfaz privada (5 en Figura 5.6). El editor de la interfaz pública es de tipo formulario y permite definir los atributos descriptivos y funcionales a partir de los conceptos que se listan en la vista `IC Ontology`. Para la interfaz privada, hemos definido un editor gráfico que permite arrastrar y soltar operaciones (de servicios web o componentes integrados) desde la vista `IC Library` al propio editor gráfico para posteriormente, conectar las operaciones seleccionadas mediante flechas que representan los aspectos de enlace. Durante el proceso de composición [E] de la Figura 5.4 se utilizan los mismos editores (4 y 5) ya que en esencia, estamos creando un componente integrado tanto en el proceso de abstracción como en el de composición. Así, la unificación de los procedimientos para el diseño de componentes integrados facilita notablemente el uso del DCI para el usuario.

El proceso de transformación [F] en la Figura 5.4 permite transformar un determinado componente integrado en un proceso WSBPEL. Al seleccionar un

componente integrado de la vista `IC Navigator`, se activa automáticamente el botón de transformación a WSBPEL [F] situado en la barra de menú superior de la vista `IC Navigator` (Figura 5.6). Al presionar dicho botón, aparece un visor con la descripción del proceso WSBPEL y la descripción WSDL correspondiente, resultado de aplicar los algoritmos de transformación (véase capítulo 4) que residen en la capa *Controller*.

Además de las vistas y editores anteriores, el DCI incluye tres vistas auxiliares para mostrar información adicional o guiar al usuario en el proceso de diseño de componentes integrados. La vista `Problems` (6) en la Figura 5.5 guía al usuario durante la creación de un nuevo componente integrado mostrando mensajes informativos, de aviso o de error según las acciones del usuario. La vista `Properties` (7) de la Figura 5.6 permite modificar ciertos atributos durante la definición de la interfaz privada de un componente integrado. Por ejemplo, en esta vista se especifican las reglas de transferencia entre parámetros de diferentes componentes integrados (atributos de enlaces) o también las propiedades (como la condición o el número de iteraciones) asociadas a ciertos patrones de selección y composición. Finalmente, la vista `Outline` (8) en la Figura 5.6 proporciona al usuario una vista jerárquica del contenido actual del editor de la interfaz privada. Ambos elementos visuales están sincronizados de tal forma que cambios en el editor de la interfaz privada se propagan directamente a la vista `Outline`, ya que ambos visualizan el mismo modelo de datos que reside en la capa *Model*.

5.3.3. Caso de aplicación

En esta sección diseñamos componentes integrados mediante el DCI retomando el escenario de control de emergencias. Siguiendo la metodología de composición (véase capítulo 4), el primero paso consistirá en el proceso de abstracción, luego el proceso de composición y finalmente, la transformación del componente integrado de monitorización a un proceso ejecutable WSBPEL.

Proceso de abstracción

El plan de actividades diseñado por nuestro protagonista Abel, esquematizado en la Figura 5.2, ha sido en realidad definido en términos de componentes integrados. Cada actividad del plan representa un posible componente integrado, por lo que el primer paso consistirá en asociar instancias concretas de servicios web a los componentes integrados imaginados por Abel. Esto es en esencia la finalidad del proceso de abstracción.

La Figura 5.7 ilustra el proceso de abstracción para nuestro escenario de control de emergencias. Los servicios web concretos están representados mediante cajas con fondo blanco. Por ejemplo, el primer componente integrado `CI-Gazetteer` engloba dos servicios web de nuestro escenario: `Pre-Emergency Plant` y `ADL Gazetteer`. El componente integrado `CI-Gazetteer` ofrece la

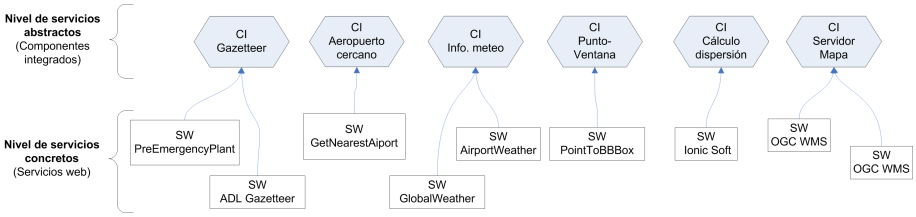


Figura 5.7: Proceso de abstracción para el caso de aplicación

operación `getCoordinates` en términos de conceptos, devolviendo una localización dado un nombre de lugar, basada en las implementaciones de los dos servicios web anteriores. La estructura interna viene definida por el patrón de selección AND-DISC que aporta cierto nivel de flexibilidad a la cadena de servicios resultante. Abel define todos los aspectos del nuevo componente integrado mediante los editores para la interfaz pública (véase Figura 5.5) y la interfaz privada (véase Figura 5.6). Una vez creado, el DCI registra automáticamente las descripciones ICML y WSDL-S en los respectivos repositorios.

El procedimiento para el diseño de componentes integrados es idéntico para los restantes componentes integrados de la Figura 5.7. Como ya se ha comentado en el capítulo 4, es importante señalar la importancia del nivel de granularidad y el de dependencia de datos para el éxito del proceso de abstracción en términos de reutilización y flexibilidad. Como ocurre con el componente integrado `CI-Gazetteer`, varias operaciones concretas con nombres distintos quedan englobadas bajo la misma operación del componente integrado. La idea consiste en aplicar la relación de descomposición (véase capítulo 4) para definir la funcionalidad de los componentes integrados a partir de la funcionalidad de los servicios web que lo forman, favoreciendo de este modo la flexibilidad porque existen varias alternativas posibles para un mismo componente integrado.

Proceso de composición

El siguiente paso para Abel consiste en el proceso de composición (de hecho, reutilización) de componentes integrados. Se trata de un proceso similar al proceso de abstracción salvo por dos matices: en primer lugar porque este proceso utiliza patrones de composición y además, cada componente integrado queda descompuesto funcionalmente mediante otros componentes integrados de nivel inferior en vez de por servicios web.

La Figura 5.8 muestra la composición y reutilización de componentes integrados para el escenario de control de emergencias. Detengámonos en la composición del componente integrado `CI-Info viento`. Tal como muestra la Figura 5.8, este componente integrado combina en secuencia tres componentes integrados de nivel inferior: `CI-Gazetteer`, `CI-Aeropuerto cercano` y `CI-Info meteo`. La Tabla 5.1 lista los componentes integrados involucrados en `CI-Info`

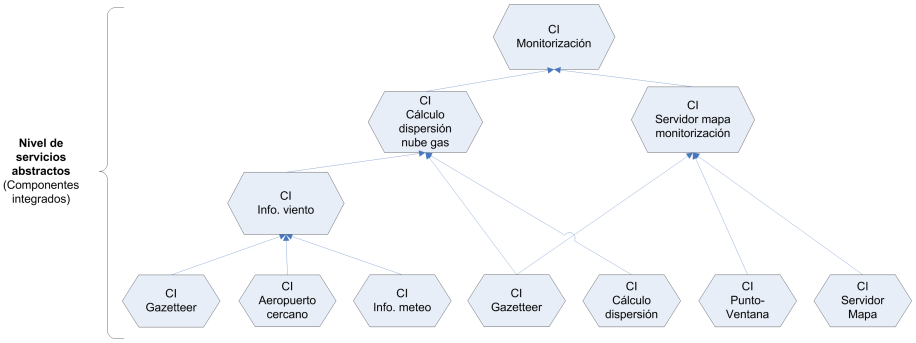


Figura 5.8: Proceso de composición para el caso de aplicación

`viento`, mostrando para cada uno de ellos el nombre de la operación, el de los parámetros de entrada y salida, y los conceptos de la ontología de dominio para anotar la operación y los parámetros.

Componente Integrado	Operación/Parámetros	Conceptos
`CI-Info viento`		
Operación	getWindInfoPlace	WindEvent
Entrada	name	City
Salida	windInfo	WindEvent
`CI-Gazetteer`		
Operación	getCoordinates	LocSpat
Entrada	name	City
Salida	coordinates	Location
`CI-Aeropuerto cercano`		
Operación	getNearestAirport	Airport
Entrada	point	Location
Salida	code	icaoCode
`CI-Info meteo`		
Operación	getWindInfo	WindEvent
Entrada	code	icaoCode
Salida	windInfo	WindEvent

Tabla 5.1: Descripción de los componentes integrados involucrados en la definición del componente integrados `CI-Info viento`

Para definir el componente integrado `CI-Info viento`, el primer paso consiste en la definición de la interfaz pública dicho componente integrado, cuya operación `getWindInfoPlace` devuelve información del viento (magnitud y vector de dirección) dada un nombre de lugar. Abel reutiliza tres componentes

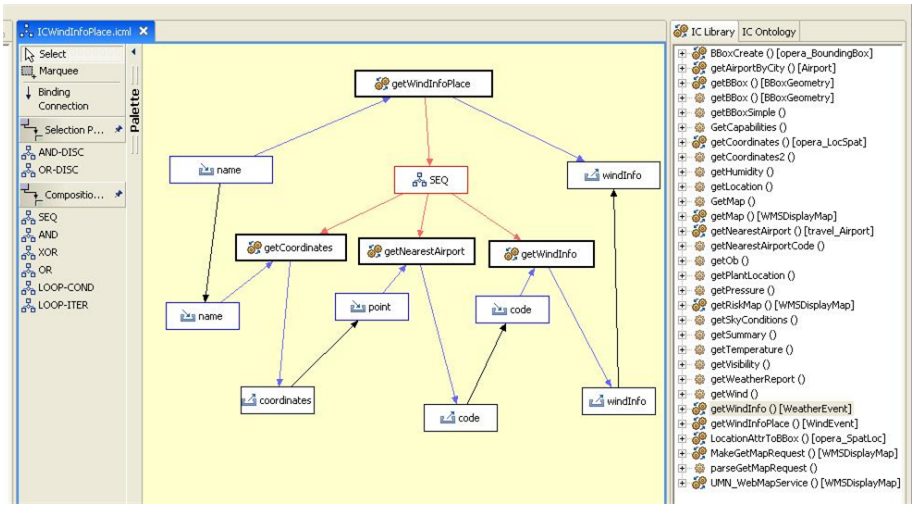


Figura 5.9: Vista del DCI definiendo la interfaz privada del componente integrado `CI-Info viento`

integrados creados anteriormente para la definición de la interfaz privada. La Figura 5.9 ilustra una vista del DCI en el momento de la definición de la interfaz privada del componente integrado `CI-Info viento`. En la parte izquierda aparece la paleta para los patrones de selección y composición. El editor gráfico para la interfaz privada aparece en el centro mientras que la librería de operaciones disponibles se lista en la vista `IC Library` en la parte derecha de la Figura 5.9. El editor gráfico permite reconocer fácilmente los diferentes objetos visuales que aparecen (patrones, parámetros, componentes integrados y servicios web) porque cada uno de ellos tiene diferentes colores. Cajas con bordes rojos representan patrones, cajas con bordes azules son parámetros de entrada y salida, y cajas con bordes negros son operaciones de componentes integrados. Manteniendo la misma semántica de colores, las flechas rojas representan la estructura definida por el patrón, las flechas azules unen parámetros con su operación, mientras que las flechas negras definen las reglas de transferencia entre parámetros, es decir, los atributos de enlace del componente integrado a definir.

Por último, cabe mencionar que la semántica introducida en la interfaz pública tiene incidencia durante la definición de la interfaz privada. Cuando se añaden operaciones de componentes integrados al editor gráfico de la interfaz privada, se establecen de forma automática los flujos de datos entre parámetros atendiendo a su anotación semántica, es decir, si los parámetros se refieren al mismo concepto. Por ejemplo, la Figura 5.9 muestra la operación `getCoordinates` del componente integrado `CI-Gazetteer`. Esta operación tiene un parámetro de entrada `name` anotada con el concepto `City`, el cual

coincide con el concepto para el parámetro de entrada `name` de la operación `getWindInfoPlace` del componente integrado que estamos definiendo (véase Tabla 5.1). En este caso, se crea automáticamente un atributo de enlace (flecha negra) entre ambos parámetros de entrada, ya que los conceptos coinciden. En ningún caso se trata de un proceso automático de inferencia semántica, sino que pretende guiar y facilitar el trabajo del usuario o diseñador.

Proceso de transformación

La tarea más sencilla, desde el punto de vista del usuario, consiste en la generación del proceso WSBPEL ejecutable. Una vez ha sido creado el componente integrado `CI-Monitorización`, tan solo hay que iniciar la acción de transformación para que el DCI muestre el código WSBPEL resultante en la vista `BPEL Viewer`. La Figura 5.10 muestra la representación WSBPEL del componente integrado `CI-Info viento` en la vista `BPEL Viewer` del DCI. Lógicamente, el código generado es bastante extenso por la verbosidad inherente del lenguaje WSBPEL. El lector interesado puede inspeccionar en el Anexo B tanto el código WSBPEL íntegro para el componente integrado `CI-Info viento` como la descripción WSDL asociada a dicho proceso. Como se aprecia en la Figura 5.10, ambas descripciones se muestran en la misma ventana pero en tabuladores separados.

Como nota final, a lo largo de esta capítulo nos hemos basado en el escenario de control de emergencias para mostrar las características del DCI. Sin embargo, creemos que nuestra aproximación basada en componentes integrados es bastante general para poder ser aplicada con éxito en otros contextos. De hecho, la capacidad de reutilizar composiciones existentes permite a usuarios expertos sacar partido de experiencias pasadas para resolver problemas similares en otros dominios [93]. Por citar únicamente dos ejemplos, otros posibles escenarios donde aplicar nuestra aproximación podrían ser la estimación del daño producido por fuertes tormentas en áreas boscosas [78], o la producción de mapas indicando la disponibilidad de recursos hídricos en entornos de alta montaña (proyecto AWARE[5]).

5.4. Resultados

La evaluación de nuestra aproximación es más o menos una cuestión subjetiva, en cuanto que los criterios de evaluación no son cuantificables, ya que tratamos con indicadores cualitativos (complejidad, flexibilidad, reutilización, etc.). El siguiente estudio consiste en establecer en primer lugar unos criterios comparativos en el contexto de la composición de servicios web, para luego discutir los resultados obtenidos de nuestra aproximación con respecto al lenguaje de composición WSBPEL en función de dichos criterios.

[5] `http://www.aware-eu.info`

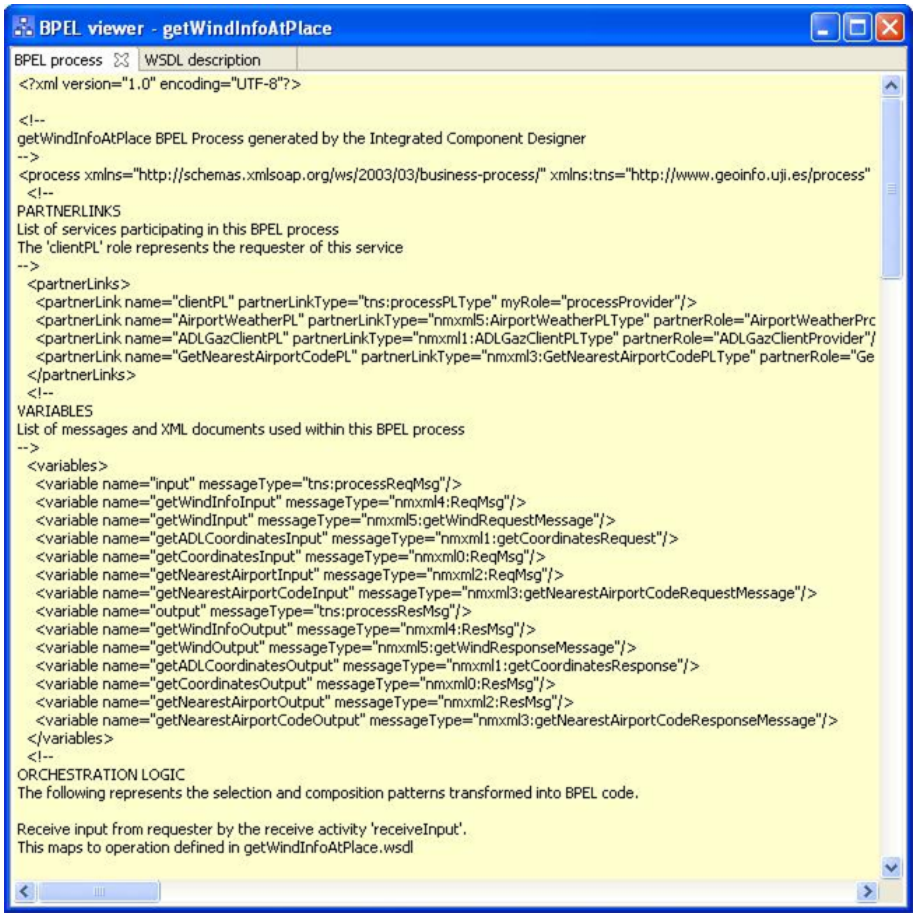


Figura 5.10: Vista del DCI mostrando el código WSBPEL para el componente integrado `CI-Info viento`

5.4.1. Criterios de comparación

Existen numerosos estudios comparativos sobre aproximaciones centradas en la composición de servicios web planteados desde perspectivas diferentes como por ejemplo, la comparación de aproximaciones estáticas *versus* dinámicas, o automáticas *versus* manuales [30, 46, 100, 129, 139]. Sin embargo, todavía no hay un consenso bien establecido acerca de los requisitos que debe satisfacer cualquier aproximación para la composición de servicios web, por lo que resulta todavía una tarea difícil establecer una marco estandarizado a este respecto. Por ejemplo, algunos autores proponen los criterios de coordinación, propiedades transaccionales, de contexto, modelo de conversión o monitorización de la ejecución [46]. Mientras otros se inclinan por la conectividad de los servi-

cios, la existencia de propiedades no funcionales para la calidad de servicio o la escalabilidad de la composición [100]. Como sucede en muchos casos, resulta casi imposible establecer cierto consenso por la proliferación excesiva de *pseudo-estándares* específicos acerca de un mismo asunto enfatizando diferentes características [154].

Para establecer los criterios mínimos de comparación en este estudio, partimos del hecho que desde la perspectiva de un desarrollador, la posibilidad de reutilizar servicios es una característica muy deseable, mientras que un usuario espera que la composición de servicios le ofrezca una acceso sencillo a una gran cantidad de servicios complejos. De este forma, hemos seleccionado ciertos criterios propuestos en [100] junto con los objetivos planteados en esta tesis para el desarrollo de nuestra aproximación. Desde nuestro punto de vista, una aproximación debe ser evaluada atendiendo a los siguientes criterios: conectividad, propiedades no funcionales y escalabilidad, todos ellos prestados de [100] más los criterios de simplicidad o reducción de la complejidad, flexibilidad y reutilización.

En principio, cualquier aproximación debería ofrecer *conectividad* entre los propios servicios que forman una composición a nivel de conexión entre los mensajes de entrada y salida. Además, resulta interesante disponer de ciertas *propiedades no funcionales* que determinen por ejemplo la calidad de un servicio o su coste. A medida que una composición aumenta de complejidad, es deseable que la aproximación para la composición de servicios ofrezca la *escalabildad* suficiente para hacer frente al número creciente de servicios involucrados. Sin embargo sin oponerse al criterio de *simplicidad*, ya que debería resultar igual de sencillo para un usuario componer diez que cien servicios web. También, una aproximación para la composición debería ofrecer composiciones *flexibles* que fueran capaces de servir el mejor servicio de entre un conjunto de servicios con la misma funcionalidad. Finalmente, según la definición en [160], una aproximación debería permitir utilizar y *reutilizar* composiciones de forma indistinguible para el usuario.

5.4.2. Discusión

Una vez especificados los criterios de comparación, procedemos con la comparación entre el lenguaje de composición de servicios web WSBPEL [105] y nuestra aproximación presentada a lo largo de esta tesis, con respecto a los seis criterios anteriores. La Tabla 5.2 resume los resultados obtenidos.

Conectividad

Ambas aproximaciones ofrecen conectividad entre los servicios que forman una composición ya que ambas producen como resultado un proceso WSBPEL [100]. Los servicios web se conectan mediante flujos de mensajes de entrada y salida por los `port types` de los servicios web, aunque a más alto

Criterios	WSBPEL	Aprox. CI
Conectividad de servicios	Alta	Alta
Propiedades no funcionales	N/A	Baja
Escalabilidad de la composición	Media	Media
Complejidad	Media	Baja
Flexibilidad	Baja	Media
Reutilización	Baja	Alta

Tabla 5.2: Resultados comparativos

nivel los servicios web son modelados de diferente modo, bien como actividades básicas en WSBPEL o como componentes integrados en nuestra aproximación.

Propiedades no funcionales

WSBPEL no proporciona por si solo mecanismos para especificar propiedades no funcionales relacionadas con la calidad del servicio como coste, seguridad o indicadores de rendimiento [105, 100]. Sin embargo, combinado con las especificaciones conocidas como Web Services Transactions[6], es posible describir procesos WSBPEL con protocolos para coordinar acciones y propagar contextos entre servicios (WS-Coordination) así como definir transacciones de corta duración (WS-AtomicTransaction) y de larga duración (WS-BussinesActivity).

En una arquitectura SOA es imprescindible que los servicios estén descritos adecuadamente [69]. Pero no utilizando únicamente especificaciones para definir interfaces, sino también metadatos para describirlos correctamente [26]. Se necesita, pues, una descripción más rica y exacta de los servicios, incluyendo tanto características funcionales como no funcionales consideradas como restricciones de la funcionalidad de un servicio [116]. Un primer paso hacia la definición de las propiedades no funcionales de un servicio consistiría, al menos, en describir sin ambigüedades que hace. Es decir, un servicio debería expresar claramente qué significa la función que ofrece. En este sentido, nuestra aproximación, además de ofrecer la funcionalidad sintáctica como en WSBPEL, tiene en cuenta también aspectos semánticos de forma que la funcionalidad de un componente integrado queda expresada sin ambigüedades. Como se explicó en el capítulo 3, los aspectos funcionales de un componente integrado están anotados mediante conceptos de ontologías de dominio. Básicamente, un usuario puede entender directamente lo que hace un componente integrado simplemente examinando los conceptos asociados a la función y sus parámetros, los cuales describen con mayor precisión la funcionalidad ofrecida que únicamente de un modo sintáctico.

[6] `http://www-128.ibm.com/developerworks/library/specification/ws-tx/`

Escalabilidad

Desde la perspectiva de un usuario, la tarea de componer dos servicios web es completamente diferente a la tarea de componer diez o cien servicios web, siendo más habitual el segundo escenario en el mundo real que el primero. Por este motivo, la escalabilidad es un aspecto crítico para la composición de servicios, ya que resulta interesante que una aproximación sea capaz de tratar adecuadamente el número creciente de servicios involucrados [100].

Sin embargo, responder directamente a la pregunta de si una aproximación es escalable o no puede resultar engañoso si no tenemos en cuenta el nivel de granularidad de los servicios involucrados en la composición. Ya en el capítulo 3 definíamos la granularidad como el grado de refinamiento funcional de un servicio. Para un diseñador, la granularidad de un servicio indica qué capacidad funcional debe ofrecer un servicio para que se ajuste a sus necesidades. Los servicios con granularidad baja (*fine-grained*) ofrecen funcionalidades limitadas o intercambian pequeñas cantidades de dados. En cambio, los servicios de granularidad alta (*coarse-grained*) encapsulan en una única interfaz más capacidad funcional, pudiendo intercambiar gran cantidad de datos en una única llamada. Como resultado, es completamente diferente escalar servicios *coarse-grained* que servicios *fine-grained*. En el primer caso, suelen bastar pocos servicios para conseguir la tarea mientras que en el segundo suelen necesitarse muchos más servicios para construir la composición deseada. En definitiva, podemos examinar teóricamente la escalabilidad de una aproximación dada pero estará influenciada, en su vertiente práctica, por el nivel de granularidad de los servicios que van a ser compuestos.

En WSBPEL, dejando a parte el nivel de granularidad de los servicios, a medida que aumenta el número de servicios en una composición el fichero XML correspondiente (con extensión bpel) también crece, dificultando la visión global del proceso ya que el usuario trata con muchos servicios web al mismo tiempo. Sin embargo, WSBPEL soporta la definición de composiciones de servicios web de forma recursiva porque un proceso WSBPEL queda expuesto a su vez como un servicio web [75]. Esta composición modular es posible por la combinación de un par de actividades –`receive` y `reply`– que permiten recibir el mensaje de entrada de un proceso y responder con el mensaje de salida respectivamente. Al trasladar estas interacciones a una descripción WSDL, el proceso WSBPEL se convierte en un servicio web.

Nuestra aproximación consigue buena escalabilidad mediante el concepto de componente integrado. Como ya se ha explicado, el proceso de composición combina incrementalmente componentes integrados, donde en cada paso el usuario interactúa al mismo tiempo con unos pocos componentes integrados que a su vez encapsulan la complejidad del proceso. La composición gradual facilita la visión de conjunto que tiene el usuario sobre la composición que está realizando.

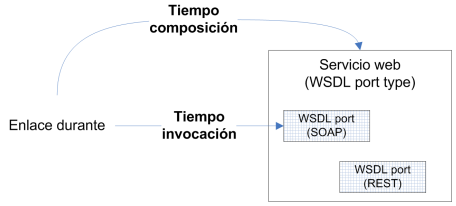
Complejidad

Nuestra aproximación propone un mecanismo de composición menos complejo que WSBPEL debido fundamental al concepto de componente integrado y al conjunto unificado de patrones. Un componente integrado es la pieza básica reutilizable que actúa de modelo de componente consiguiendo que la composición de componentes integrados produzca únicamente componentes integrados. En WSBPEL, las actividades básicas (`invoke`, `receive` y `reply`), que representan invocaciones a servicios web, representan el modelo de componente [7]. La composición de actividades básicas mediante actividades estructuradas (por ejemplo `sequence` y `flow`) produce procesos que son de distinta naturaleza que las actividades básicas. Por lo tanto, en WSBPEL no existe un único tipo de modelo de componente, aumentando en consecuencia la complejidad de la aproximación.

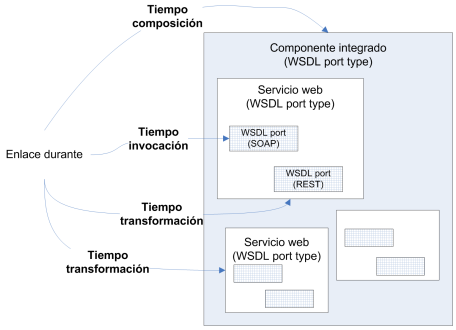
Respecto a los patrones, WSBPEL presenta una amplio abanico de actividades estructuradas (patrones), encargadas de la composición de otras actividades básicas o estructuradas. En la vertiente negativa, se trata de un lenguaje complejo porque ofrece actividades estructuradas redundantes para modelar cierto workflow dependiendo del estilo seleccionado (WSFL o XLANG), e incluso permite combinar diferentes actividades estructuradas de ambos estilos [121, 161]. Sin embargo, nuestro objetivo en este trabajo ha consistido en alcanzar un número mínimo pero adecuado de patrones sin solapamientos entre ellos que permita modelar cualquier workflow. En este sentido, la simplicidad de los patrones es un factor que reduce la complejidad de una aproximación.

Flexibilidad

El concepto de flexibilidad puede variar dependiendo tanto del contexto como del momento en que tiene lugar. Por ejemplo, WSBPEL posibilita que en tiempo de ejecución una misma operación de un servicio pueda ser invocada utilizando diferentes implementaciones de la misma operación [75]. La Figura 5.11(a) muestra un servicio web que contiene una única operación implementada mediante diferentes mecanismos de comunicación como por ejemplo SOAP y REST [48]. WSBPEL facilita flexibilidad en la implementación de una operación mediante la separación entre interfaces (`port types`) y mecanismos de comunicación (`ports`), ya que únicamente especificando la interfaz concreta en tiempo de composición el mecanismo de comunicación se establecerá en tiempo de ejecución, permitiendo que sea invocada bien la operación utilizando SOAP o bien la operación utilizando REST. Sin embargo, una vez especificados en tiempo de diseño los atributos de una actividad `invoke` de WSBPEL como el nombre de la interfaz (`port type`) y el de la operación a invocar (`operation`), ya no es posible modificarlos en tiempo de ejecución. Luego la flexibilidad en WSBPEL está limitada a intercambiar entre los diferentes `ports` (misma operación bien vía SOAP o REST) de un servicio web [122]. Esto implica que si todas las implementaciones de un servicio web fallan, no es posible continuar



(a) WSBPEL



(b) Componentes integrados

Figura 5.11: Influencia del momento de enlace en la flexibilidad

con la ejecución de la composición porque no existen servicios web alternativos definidos para el servicio web que ha fallado.

Yendo un paso más allá, en el capítulo 3 definíamos la flexibilidad como la capacidad de servir el mejor de entre un conjunto de servicios que ofrecen una funcionalidad similar. La Figura 5.11(b) muestra como mediante componentes integrados se establece en tiempo de composición un enlace con la operación del componente integrado. Sin embargo, este hecho no limita esta operación a una instancia concreta de un servicio web como ocurre en WSBPEL. Hasta el momento de la transformación de un componente integrado a WSBPEL, no se establecen los enlaces correspondientes con todas las operaciones del conjunto de servicios web que forman el componente integrado. De esta forma, nuestra aproximación permite diseñar composiciones de servicios web mucho más flexibles porque cada operación de un componente integrado no está ligada a un único servicio web en concreto en tiempo de composición, sino que está asociada a un conjunto de servicios web que ofrecen funcionalidades similares. Si falla la operación de un servicio web, el patrón de selección del componente integrado permite continuar con la operación de otro servicio de entre el conjunto de servicios web candidatos.

Reutilización

El criterio de reutilización está ligado al criterio de escalabilidad en el sentido que también depende del grado de granularidad de los servicios web involucrados. Desde la perspectiva del diseñador, surge la pregunta sobre qué tipo de servicios, *fine-grained* o *coarse-grained*, son más reutilizables. En principio, para responder a esta pregunta, hay que identificar previamente si un determinado servicio va a ser utilizado una única vez o está pensado para ser utilizado en múltiples ocasiones. En el primer caso, nos encontramos en situaciones en que un diseñador construye una composición para una tarea específica, sin pensar en que pueda ser reutilizada con posterioridad. Podríamos identificar este tipo de composiciones como de "usar y tirar". En el segundo caso, el diseñador está pensando en construir composiciones que pueden servir en otras composiciones incluso en otros contextos. En este caso, servicios que lleven a cabo funcionalidades reducidas, como los *fine-grained*, pueden ser bastante reutilizables, es decir, pueden ser incorporados con facilidad en otros escenarios donde su funcionalidad sea requerida. Sin embargo, gran cantidad de este tipo de servicios puede producir sistemas saturados e ineficientes. Por este motivo, algunos analistas[7] sobre servicios web proponen como mejor opción, una colección reducida de servicios *coarse-grained* que puedan ser reutilizados en múltiples escenarios.

Nuestra aproximación permite ambas posibilidades. Por un lado un usuario puede construir componentes integrados con funcionalidades específicas que pueden ser fácilmente reutilizados por otros componentes integrados. Pero, al mismo tiempo, nuestra aproximación también permite reutilizar componentes integrados *coarse-grained*, cuando se reutiliza un componente integrado compuesto a su vez de otros tantos componentes integrados, encapsulando la funcionalidad de todos los componentes integrados involucrados. De esta forma, resulta sencillo proporcionar una librería de componentes integrados *coarse-grained* para que sean reutilizados por usuarios. Pero, a su vez, estos componentes integrados están reutilizando otros componentes integrados *fine-grained*, que pueden estar registrados en repositorios privados no accesibles directamente por los usuarios. Este comportamiento es conocido como servicio translúcido (*translucent*) en la terminología ISO 19119 [70], donde los usuarios interaccionan con un servicio mediador que controla la cadena de servicios subyacentes. Para un usuario, una composición es un único componente integrado, el cual integra la complejidad del proceso en forma de combinaciones de otros componentes integrados, proporcionando la opción óptima entre una composición completamente automática y una completamente controlada por el usuario [6]. En definitiva, nuestra aproximación proporciona reutilización a diferentes niveles en función del criterio del diseñador.

[7] `http://www.zapthink.com/report.html?id=ZAPFLASH-200639`

5.5. Resumen

En la primera parte de este capítulo hemos presentado la herramienta DCI, el Diseñador de Componentes Integrados, junto con la arquitectura basada en el paradigma MVC [80]. El DCI ha sido desarrollado en línea con el concepto de componente integrado (véase capítulo 3) y la metodología de composición (véase capítulo 4), teniendo en cuenta además ciertos requisitos de diseño como la facilidad de uso, que sea intuitivo y haciendo indistinguible la utilización o reutilización de componentes integrados.

Finalmente, en la segunda parte hemos presentado un estudio comparativo de nuestra aproximación con respecto al lenguaje de composición de servicios web WSBPEL. Los resultados cualitativos obtenidos han mostrado como nuestra aproximación proporciona mejores niveles de flexibilidad y reutilización comparado con WSBPEL en tiempo de diseño. Además, la complejidad de nuestra aproximación se reduce gracias principalmente a la introducción del concepto de componente integrado como modelo de componente y al conjunto unificado de patrones propuesto. Sin embargo, no se debe interpretar nuestra aproximación como una alternativa a WSBPEL, sino mas bien como una aportación adicional que trata de solventar las carencias de este lenguaje. Por lo tanto, procesos WSBPEL siguiendo nuestra aproximación, diseñados mediante componentes integrados, resultan mucho más flexibles y reutilizables.

Capítulo 6

Conclusiones

Este capítulo resume el trabajo desarrollado y presenta las contribuciones más importantes alcanzadas en esta investigación. También plantea algunas direcciones para futura investigación y enumera las publicaciones más relevantes derivadas de esta tesis.

6.1. Consideraciones generales

El trabajo de esta tesis ha sido motivado por la necesidad de disponer de mecanismos más sencillos que den soporte a un modelo de composición de servicios web más flexible, y al mismo tiempo explotando la reutilización, para ofrecer al usuario aplicaciones a medida. En consecuencia, el objetivo ha sido facilitar el desarrollo de aplicaciones web mediante composiciones reutilizables y flexibles.

Hemos abordado principalmente el concepto de reutilización de software en diferentes contextos, como por ejemplo, las relaciones y diferencias entre los componentes software y los servicios web, o la necesidad de adaptar las características inherentes de los componentes software al contexto de los servicios web. También, hemos mostrado la influencia que ejercen los workflows en el proceso de composición de servicios web. Hemos visto como los patrones de workflow son esenciales para la especificación del flujo de control entre servicios web. Mediante un análisis exhaustivo y en varias fases, hemos adaptado los patrones de workflow tradicionales de los WfMS tanto al contexto de los servicios web en general como al contexto de los componentes integrados en particular.

Una parte importante de esta tesis se ha centrado en la definición de componente integrado como pieza básica para conseguir composiciones reutilizables. La descripción de un componente integrado ha sido dividida en diferentes aspectos, enfatizando el hecho de capturar un nivel adecuado de detalle de un componente integrado con el fin de maximizar su reutilización. A lo largo del capítulo 3 se han abordado diferentes cuestiones relativas a un componente integrado como por ejemplo la descripción de las interfaces pública y privada, la anotación semántica de la interfaz funcional mediante conceptos definidos en ontologías o, la integración y encapsulación de los patrones en la descripción de un componente integrado.

Hemos propuesto una metodología para la creación, composición y transformación de componentes integrados, definiendo un proceso conceptual para cada una de estas funciones. Cada proceso conceptual ha sido descrito mediante un conjunto de fases y mecanismos necesarios para llevarlo a cabo de una forma práctica. Los diferentes procesos se relacionan entre sí mediante una arquitectura en niveles de abstracción, en la cual hemos definido una relación de descomposición entre componentes integrados de niveles adyacentes.

Finalmente, para dar soporte al diseño de componentes integrados, hemos desarrollado una herramienta software denominada Diseñador de Componentes Integrados (DCI), encargada de trasladar a la práctica el concepto de componentes integrado y su metodología de composición.

6.2. Contribuciones

La contribución más importante de esta tesis ha consistido en la definición de un modelo de composición de servicios web basado en componentes integrados que hace uso de la reutilización como mecanismo de composición. Este modelo responde afirmativamente a la pregunta planteada en la introducción de esta tesis: ¿Es posible reutilizar composiciones existentes y partes de éstas para crear nuevas composiciones de servicios web?. Nuestra aproximación no excluye el uso de otras aproximaciones o modelos para la composición de servicios web. Mas bien, debe interpretarse como una aproximación complementaria a otras para mejorar la reutilización de composiciones. De hecho, necesita de soluciones ya existentes para poder aprovechar su potencial. Pongamos el caso de WSBPEL, un lenguaje para la composición de servicios web pero que no presta bastante atención a la reutilización de composiciones de servicios web. La combinación de nuestra aproximación con WSBPEL ha demostrado mejorar las capacidades iniciales del lenguaje WSBPEL en términos de reutilización y flexibilidad. Nuestra aproximación aporta un diseño diferente para la creación de composiciones de servicios web, mientras que WSBPEL sigue siendo útil como lenguaje de ejecución de composiciones de servicios web. Por lo tanto, procesos WSBPEL siguiendo nuestra aproximación, diseñados mediante componentes integrados, resultan mucho más flexibles y reutilizables.

El desarrollo de la herramienta DCI nos ha permitido experimentar en dos vertientes distintas. En primer lugar, hemos puesto en práctica nuestra aproximación para demostrar que la reutilización es un mecanismo perfectamente válido, no solo en los sistemas basados en componentes sino también en un contexto poco explorado como el de la composición de servicios web. En segundo lugar, el DCI nos ha permitido explorar la creación de aplicación complejas encima de una infraestructura IDE. La creación de aplicaciones basadas en composiciones de servicios web sigue siendo una tarea pendiente en el desarrollo futuro de la IDE [128]. Varios autores [21] ya apuntan la necesidad de desplazar la visión actual de la IDE, centrada en los datos, a una visión mucho más amplia que abarque además servicios distribuidos geográficamente a diferentes niveles (nacional, local, etc.), buscando como último fin ofrecer la visión de una infraestructura espacial dirigida completamente por servicios. Por lo tanto, el continuo desarrollo de soluciones innovadores para la composición de servicios web en el marco de la IDE e INSPIRE, acercará cada vez más la visión de una infraestructura de servicios espaciales. Nuestro trabajo intenta contribuir un poco más al desarrollo y evolución de las infraestructuras de datos y servicios espaciales, al menos, en su faceta más tecnológica.

La reutilización de servicios web y de composiciones mediante componentes integrados no ofrece únicamente la posibilidad de reutilizar descripciones de servicios, aspecto clave en arquitecturas SOA, sino que además proporciona la capacidad de sacar provecho de otras composiciones ya desarrolladas por otros usuarios ante problemas similares. La reutilización de composiciones puede facilitar la transferencia de conocimiento entre usuarios, puesto que soluciones y experiencias previas pueden ser aplicadas a problemas similares [93].

Para conocer el verdadero alcance de una tesis, resulta interesante conocer tanto sus contribuciones como sus limitaciones. Respecto a este último aspecto, la aproximación propuesta no pretende convertirse en un modelo de reutilización de servicios web entre diferentes contextos, donde un servicio web cualquiera puede ser sustituido por otro sin importar si un servicio habla en términos de planificación urbana y otro lo hace en términos de información meteorológica. Nuestro trabajo se ciñe a un contexto en particular, en el cual expertos en el dominio han establecido una ontología que será compartida por todas las composiciones de servicios que se creen y reutilicen en dicho dominio.

También cabe destacar que, para que nuestra aproximación tenga éxito, el usuario debe conocer de antemano los servicios web que van a ser transformados en componentes integrados (proceso de abstracción). Como comentábamos en el capítulo 3, el problema general de la composición puede desglosarse a su vez en tres grandes tareas [141]: la confección de un plan de las actividades para resolver el problema; la localización de los servicios web que se ajusten a las actividades del plan; y la composición de estos servicios web de forma que interactúen entre ellos de forma adecuada. La última tarea ha sido el foco de interés de esta tesis. Por esta razón, consideramos que el usuario conoce y tiene disponibles los servicios web que necesitará durante el proceso de composición.

6.3. Direcciones futuras de investigación

El trabajo desarrollado en esta tesis representa la base inicial para establecer un modelo de composición de servicios web basado en la reutilización mucho más extenso y que cubra otros aspectos necesarios de los servicios web. Existe una gran variedad de puntos y direcciones que todavía quedan por resolver en el ámbito de los servicios web, como la semántica, seguridad, calidad de servicio, contratos y negociaciones o aspectos transaccionales [74]. A continuación, mostramos algunas líneas de trabajo que pueden mejorar considerablemente el trabajo de esta tesis.

6.3.1. Reutilización contextual

Introducíamos esta tesis con el informe difundido por la Comisión Europea acerca de las visiones de futuro de los servicios, en el cual se hacía hincapié explícitamente en facilitar la integración de servicios provenientes de diferentes contextos y dominios. Una futura línea de investigación consistiría en servicios web que sean conscientes del contexto en el que actúan. La idea es que los servicios web hagan uso de la información del contexto que les rodea, con el fin de ofrecer ciertos servicios a determinados usuarios en función, por ejemplo, de la localización física del usuario.

Normalmente, la información contextual de los servicios web estática o predefinida suele ser fácilmente procesable. Sin embargo, uno de los mayores problemas para conseguir servicios web que sean conscientes del contexto consiste en la complejidad de capturar y representar información contextual a medida. Un posible reto de futuro sería el estudio de la recuperación y diseño de información contextual en composiciones de servicios, para ofrecer composiciones de servicios web contextuales, capaces de crear *ad hoc* el contexto adecuado de una composición como resultado de los contextos de los servicios web que la forman. La reutilización contextual puede ser aplicable en muchos escenarios distintos como en servicios web para dispositivos móviles, donde la localización es un aspecto crucial junto con otros factores como el tiempo o los gustos y preferencias del usuario.

6.3.2. Búsqueda semántica de composiciones

Un complemento importante a la composición y reutilización de servicios web es el descubrimiento de servicios web. Un primer paso en esta dirección está siendo el trabajo conjunto desarrollado entre investigadores del ITC[1] en Holanda y la UJI [86, 56]. En estos trabajos hemos aportado una posible solución para resolver el problema completo de la composición de servicios web, es decir descubrimiento, composición y ejecución de servicios web, integrando dos aproximaciones independientes entre sí aunque complementarias: una para la

[1] `http://www.itc.nl`

localización de servicios a partir de un plan o consulta semántica [85], junto con nuestra aproximación basada en componentes integrados para la composición y reutilización de servicios web presentada a lo largo de esta tesis. Un resultado destacable de este trabajo conjunto es la capacidad de descubrir semánticamente componentes integrados, es decir, composiciones de servicios web listas para ser reutilizadas.

6.3.3. Estándares OGC

El OGC está inmerso en el desarrollo de la especificación Web Processing Service (WPS) [111], todavía en continua evolución puesto que se trata de un documento de trabajo. WPS especifica una interfaz de propósito general de procesos que contienen cálculos y modelos para el proceso de datos geo-espaciales. Los cálculos que ofrece un WPS pueden ser de cualquier naturaleza, relativamente sencillos como el cálculo de cierta área de bosque quemada dada una imagen de la zona afectada, o más complejos como modelos climáticos o hídricos. La idea, como en las restantes especificaciones OGC, es que cualquier cálculo pueda ajustarse al interfaz definido por la especificación WPS para ofrecer procesos de datos geo-espaciales interoperables.

Aunque WPS se encuentra todavía en sus inicios, es interesante seguir muy de cerca su evolución porque esta especificación es muy afín al trabajo de esta tesis sobre composición y reutilización. Desde la perspectiva de esta tesis, vale la pena investigar la relación entre WSBPEL y WPS en cuanto que la finalidad básica de ambas especificaciones es la invocación de servicios. Responder a las preguntas de si WPS puede integrarse en WSBPEL y qué ventaja se consigue con ello; o bien si WPS puede sustituir a WSBPEL únicamente en determinados escenarios específicos o en cualquiera. Una posible línea de futura investigación podría centrarse en el estudio de la composición de procesos WPS y su relación con el concepto de componente integrado para permitir la reutilización de composiciones de procesos WPS. Esta línea es muy atractiva para el desarrollo de aplicaciones en una IDE [21], porque incide claramente en la creación de aplicaciones orientas a servicios justo encima de una IDE ya que los procesos WPS manejan datos geo-espaciales proporcionados por una IDE.

6.4. Publicaciones derivadas

Los resultados de esta tesis han sido publicados –o se encuentran en proceso de revisión– en revistas y conferencias internacionales dentro del campo de la ingeniería web, servicios web, los sistemas de información geográfica en general y de las infraestructuras de datos espaciales.

A continuación se enumeran y describen brevemente algunas de las publicaciones más relevantes. Otras publicaciones menos relevantes derivadas también de esta tesis pueden encontrarse en el apartado de Bibliografía.

- Granell C., Gould M. "An Integrated Component Framework for Service Reuse"
  Enviado a International Journal of Information Technology and Web Engineering, 2006 ([54]).

Esta publicación concentra todos los capítulos presentados en esta tesis, haciendo hincapié en el concepto de componente integrado, la metodología de composición, la herramienta software DCI y exponiendo finalmente trabajos relacionados.

- Granell C., Gould M., Lemmens R., Wytzisk A., de By R., van Oosterom P. "Integrating semantic and syntactic descriptions for chaining geographic services"
  Enviado a IEEE Internet Computing, 2006 ([56])

- Lemmens R., Granell C., Wytzisk A., de By R., Gould M., van Oosterom P. "Semantic and syntactic service descriptions at work in geo-service chaining"
  Proc. of the 9th AGILE International Conference on Geographic Information Science. University of West Hungary, pp. 51-61, 2006 ([86])

Estos trabajos integran dos aproximaciones independientes para tratar con el descubrimiento, composición y ejecución de servicios web. La primera aproximación permite localizar servicios a partir de un plan o consulta semántica, mientra que la segunda aproximación permite componer los servicios web localizados mediante el concepto de componente integrado. La contribución más destacable de estos trabajos es la capacidad de descubrir componentes integrados mediante una consulta semántica, permitiendo la reutilización de composiciones de servicios web.

- Granell C., Gould M., Grønmo R., Skogan D. "Improving Reuse of Web Service Compositions"
  Proc. of the 6th International Conference on E-Commerce and Web Technologies. Springer LNCS 3590, pp. 358-367, 2005 ([55])

Este trabajo se centra prácticamente en el proceso de transformación, describiendo detalladamente los patrones de composición y selección mostrados durante el análisis realizado en el capítulo 3.

- Granell C., Gould M., Ramos J.F. "Service Composition for SDIs: integrated components creation"
  Proc. of DEXA Workshop 2005 (GIM 2005). IEEE CS Press, pp. 475-479, 2005 ([57])

- Poveda J., Gould M., Granell C. "Composition of e-Commerce and Geographic Information Services for Emergency Management"
  Proc. of the 3rd. International Conference on Electronic Government (EGOV 2004). Springer LNCS 3183, pp 562-563, 2004 ([126])

- Granell C., Poveda J., Gould M. "Incremental Composition of Geograhic Web Services: An Emergency Management Context"
  Proc. of the 7th AGILE Conference on Geographic Information Science, Crete University Press, pp 343-348, 2004 ([60])

Estos trabajos analizan la relación existente entre nuestra aproximación con la creación de aplicaciones basadas en servicios utilizando recursos de una IDE. Aunque se plantea y revisa el concepto de componente integrado, el foco de atención en todas ellas es un escenario práctico con el que demostrar la viabilidad de nuestra aproximación, tal como se ha presentado en el capítulo 5.

- Granell C., Poveda J., Gould M. "An Incremental Approach to Web Service Composition"
  Colombian Journal of Computation, 5(1):22-34, Junio 2004 ([59])

- Granell C., Ramos J.F. "An Object-Oriented Approach to GI Web Service Composition".
  Proc. of DEXA Workshop 2004. IEEE CS Press, pp 835-839, 2004 ([61])

- Granell C., Poveda J., Gould M. "Incremental Weak Composition and Invocation of Geographic Web Services"
  Proc. of the 2nd International Workshop on Semantic Processing of Spatial Data. Centre for Computing Research, pp 179-187, 2003 ([58])

Estos trabajos plantean las versiones iniciales del concepto de componente integrado y de la metodología de composición. Evidentemente, los conceptos presentados en esta tesis son el resultado de un proceso constante de mejora y refinamiento, gracias en gran medida a las sugerencias y comentarios proporcionados por los revisores durante las primeras etapas de esta tesis.

ANEXO A

Esquema para la representación de componentes integrados

A continuación se muestra la definición formal del Integrated Component Markup Language (ICML) escrito mediante la notación de XML Schema[1].

```
<?xml version='1.0' encoding='UTF-8'?>

<!--

XML schema for the Integrated Component Markup Language (ICML)
version 1.0

-->

<xs:schema
 targetNamespace='http://www.geoinfo.uji.es/icml'
 xmlns:xs='http://www.w3.org/2001/XMLSchema'>

  <xs:complexType name='tExtensibleBase'>
    <xs:annotation><xs:documentation>
```

[1] http://www.w3.org/XML/Schema

```
  This type is extended by component types to allow elements
  and attributes from other namespaces to be added.
  </xs:documentation></xs:annotation>
  <xs:sequence>
    <xs:annotation><xs:documentation>
    namespace=##other: any namespace other than target
    namespace.
    processContents=lax: validate where
    you can, do not worry when you cannot.
    </xs:documentation></xs:annotation>
    <xs:any maxOccurs='unbounded' minOccurs='0'
         namespace='##other' processContents='lax'>
    </xs:any>
  </xs:sequence>
  <xs:anyAttribute
    namespace='##other' processContents='lax'>
  </xs:anyAttribute>
</xs:complexType>


<!-- Root Element -->
<xs:element name='ICMLModel'>
  <xs:annotation><xs:documentation>
  Root Element: The root element of an Integrated
  Component Markup Language document.
  </xs:documentation></xs:annotation>
  <xs:complexType>
    <xs:extension base='tExtensibleBase'>
      <xs:all>
        <!-- Public Interface -->
        <xs:element maxOccurs='1' minOccurs='0'
             name='PublicInterface'>
          <xs:sequence>
            <!-- Description Section -->
            <xs:element maxOccurs='1' minOccurs='0'
               ref='DescriptionSection'></xs:element>
            <!-- Functionality Section -->
            <xs:element maxOccurs='1' minOccurs='0'
               ref='FunctionalitySection'></xs:element>
          </xs:sequence>
        </xs:element>
        <!-- Private Interface -->
        <xs:element maxOccurs='1' minOccurs='0'
           name='PrivateInterface'>
```

```
          <xs:sequence>
            <!-- Structure Section -->
            <xs:element maxOccurs='1' minOccurs='0'
                 ref='StructureSection'></xs:element>
            <!-- Binding Section -->
            <xs:element maxOccurs='1' minOccurs='0'
               ref='BindingSection'></xs:element>
          </xs:sequence>
        </xs:element>
      </xs:all>
    </xs:extension>
  </xs:complexType>
</xs:element>


<!-- Description Section -->
<xs:element name='DescriptionSection'>
  <xs:annotation><xs:documentation>
  DescriptionSection element: This element represents the
  descriptive aspects included in an integrated component.
  </xs:documentation></xs:annotation>
  <xs:complexType>
    <xs:extension base='tExtensibleBase'>
      <xs:all>
        <xs:element maxOccurs='1' minOccurs='1'
           name='ID' type='xs:string'></xs:element>
        <xs:element maxOccurs='1' minOccurs='1'
                    name='AccessPoint' type='xs:string'>
        </xs:element>
        <xs:element maxOccurs='1' minOccurs='0'
                    name='Description' type='xs:string'>
        </xs:element>
      </xs:all>
    </xs:extension>
  </xs:complexType>
</xs:element>


<!-- Functionality Section -->
<xs:element name='FunctionalitySection'>
  <xs:annotation><xs:documentation>
  FunctionalitySection element: This element represents
  the functionality or capability offered by an
  integrated component. The functionality aspects of an
```

```
  integrated component consists of input and output lists
  of user-defined parameters and the function name.
  </xs:documentation></xs:annotation>
  <xs:complexType>
    <xs:extension base='tExtensibleBase'>
      <xs:all>
        <xs:element maxOccurs='1' minOccurs='1'
             ref='Operation'></xs:element>
        <xs:element maxOccurs='1' minOccurs='1'
             ref='Inputs'></xs:element>
        <xs:element maxOccurs='1' minOccurs='1'
           ref='Outputs'></xs:element>
      </xs:all>
    </xs:extension>
  </xs:complexType>
</xs:element>


<!-- Structure Section -->
<xs:element name='StructureSection'>
  <xs:annotation><xs:documentation>
 StructureSection element: A structure section
 contains a list of contained components either
 integrated components or web services. This
 section also includes a concrete pattern instance
 defining how those contained components are combined.
 </xs:documentation></xs:annotation>
  <xs:complexType>
    <xs:extension base='tExtensibleBase'>
      <xs:all>
        <xs:element maxOccurs='1' minOccurs='1'
           ref='Pattern'></xs:element>
        <xs:element maxOccurs='1' minOccurs='1'
           name='ContainedComponents'>
          <xs:sequence>
            <xs:element maxOccurs='unbounded' minOccurs='0'
               ref='ContainedComponent'>
            </xs:element>
          </xs:sequence>
        </xs:element>
      </xs:all>
    </xs:extension>
  </xs:complexType>
</xs:element>
```

```
<!-- Binding Section -->
<xs:element name='BindingSection'>
  <xs:annotation><xs:documentation>
  BindingSection element: A data flow binding is a linking
  between a source and a target parameter in the private
  interface of an integrated component. It is possible to
  specify four kinds of bindings (input, output, internal,
  constant) depending on either the source and target
  parameters belong to the composition or to contained
  components. Additionally, if no linking bindings are
  defined in a given parameter, it is possible to
  assign it a constant value.
  </xs:documentation></xs:annotation>
  <xs:complexType>
    <xs:extension base='tExtensibleBase'>
      <xs:sequence>
        <xs:element maxOccurs='unbounded' minOccurs='0'
           ref='Binding'></xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexType>
</xs:element>


<xs:element name='Object'>
  <xs:annotation><xs:documentation>
  Object element: Any object with an ID. If not specified,
   all other elements extend this basic one, as most
   elements in an ICML document must be uniquely identifiable.
  </xs:documentation></xs:annotation>
  <xs:complexType>
    <xs:extension base='tExtensibleBase'>
      <xs:attribute name='id' type='xs:ID'
       use='required'></xs:attribute>
    </xs:extension>
  </xs:complexType>
</xs:element>


<xs:element name='VisualObject'>
  <xs:annotation><xs:documentation>
  VisualObject element: It represents any elemental object
```

```
  with specific position (x,y) and size (width, height).
  </xs:documentation></xs:annotation>
  <xs:complexType>
    <xs:extension base='Object'>
      <xs:attribute name='x' type='xs:integer'
       use='required'></xs:attribute>
      <xs:attribute name='y' type='xs:integer'
       use='required'></xs:attribute>
      <xs:attribute name='width' type='xs:integer'
       use='required'></xs:attribute>
      <xs:attribute name='height' type='xs:integer'
       use='required'></xs:attribute>
    </xs:extension>
  </xs:complexType>
</xs:element>


<xs:element name='SemanticFeatures'>
  <xs:annotation><xs:documentation>
  SemanticFeatures element: Any visual object with
  an specific name, ontology namespace and resource,
  and optionally, a constant value.
  </xs:documentation></xs:annotation>
  <xs:complexType>
    <xs:extension base='VisualObject'>
      <xs:all>
        <xs:element maxOccurs='1' minOccurs='1'
           name='Name' type='xs:string'>
        </xs:element>
        <xs:element maxOccurs='1' minOccurs='1'
         name='OntologyResource' type='xs:string'>
        </xs:element>
        <xs:element maxOccurs='1' minOccurs='1'
             name='OntologyNamespace' type='xs:string'>
        </xs:element>
        <xs:element maxOccurs='1' minOccurs='0'
           name='ConstantValue' type='xs:string'>
        </xs:element>
      </xs:all>
    </xs:extension>
  </xs:complexType>
</xs:element>
```

```
<xs:element name='ComponentFeatures'>
  <xs:annotation><xs:documentation>
  ComponentFeatures element: Any visual object
  with a specific name, an URL indicating the
  access point and the component's interface.
  </xs:documentation></xs:annotation>
  <xs:complexType>
    <xs:extension base='tExtensibleBase'>
      <xs:all>
        <xs:element maxOccurs='1' minOccurs='1'
           name='Name' type='xs:string'>
        </xs:element>
        <xs:element maxOccurs='1' minOccurs='1'
           name='AccessPoint' type='xs:string'>
        </xs:element>
        <xs:element maxOccurs='1' minOccurs='1'
             name='Interface' type='xs:string'>
        </xs:element>
      </xs:all>
    </xs:extension>
  </xs:complexType>
</xs:element>


<!-- Operation -->
<xs:element name='Operation'>
  <xs:annotation><xs:documentation>
  Operation element: This element represents the functionality
  or capability offered publicly by an integrated component.
  </xs:documentation></xs:annotation>
  <xs:complexType>
    <xs:extension base='VisualObject'>
      <xs:sequence>
        <xs:element maxOccurs='1' minOccurs='0'
             ref='Features'></xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexType>
</xs:element>


<!-- Inputs -->
<xs:element name='Inputs'>
  <xs:annotation><xs:documentation>I
```

```
  nputs element: input parameters list contained in a given
  integrated component.
  </xs:documentation></xs:annotation>
  <xs:complexType>
    <xs:extension base='tExtensibleBase'>
      <xs:attribute name='msg' type='xs:string'
         use='required'></xs:attribute>
      <!-- Message name -->
      <xs:sequence>
        <xs:element maxOccurs='unbounded' minOccurs='0'
         ref='Input'></xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexType>
</xs:element>


<!-- Input -->
<xs:element name='Input'>
  <xs:annotation><xs:documentation>
  Input element: An Input represents a concrete
  input parameter.It is also annotated semantically.
  </xs:documentation></xs:annotation>
  <xs:complexType>
    <xs:extension base='tExtensibleBase'>
      <xs:sequence>
        <xs:element maxOccurs='1' minOccurs='0'
           ref='SemanticFeatures'></xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexType>
</xs:element>


<!-- Outputs -->
<xs:element name='Outputs'>
  <xs:annotation><xs:documentation>
  Outputs element: output parameters list contained in a
  given integrated component.
  </xs:documentation></xs:annotation>
  <xs:complexType>
    <xs:extension base='tExtensibleBase'>
      <xs:attribute name='msg' type='xs:string'
       use='required'></xs:attribute>
```

```
      <!-- Message name -->
      <xs:sequence>
        <xs:element maxOccurs='unbounded' minOccurs='0'
         ref='Output'></xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexType>
</xs:element>


<!-- Output -->
<xs:element name='Output'>
  <xs:annotation><xs:documentation>
  Output element: An Output represents a concrete
  output parameter. It is also annotated semantically.
  </xs:documentation></xs:annotation>
  <xs:complexType>
    <xs:extension base='VisualObject'>
      <xs:sequence>
        <xs:element maxOccurs='1' minOccurs='0'
         ref='SemanticFeatures'></xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexType>
</xs:element>


<!-- Pattern -->
<xs:element name='Pattern'>
  <xs:annotation><xs:documentation>
  Pattern element: A Pattern is a visual representation
  of the control flow applied to the contained components
  in an integrated component.
  </xs:documentation></xs:annotation>
  <xs:complexType>
    <xs:extension base='VisualObject'>
      <xs:attribute name='type' type='PatternType'
       use='required'></xs:attribute>
      <xs:attribute name='cond' type='xs:string'>
      </xs:attribute>
      <xs:attribute name='finalCounter' type='xs:string'>
      </xs:attribute>
    </xs:extension>
  </xs:complexType>
```

```
  </xs:element>


<!-- Pattern type: selection or composition-->
  <xs:simpleType name='PatternType'>
    <xs:restriction base='xs:string'>
      <xs:enumeration value='selection'></xs:enumeration>
      <xs:enumeration value='composition'></xs:enumeration>
    </xs:restriction>
  </xs:simpleType>


<!-- Pattern instances: only for informative purpose-->
  <xs:simpleType name='PatternValue'>
    <xs:restriction base='xs:string'>
      <xs:enumeration value='AND-DISC'></xs:enumeration>
      <xs:enumeration value='OR-DISC'></xs:enumeration>
      <xs:enumeration value='SEQ'></xs:enumeration>
      <xs:enumeration value='AND'></xs:enumeration>
      <xs:enumeration value='OR'></xs:enumeration>
      <xs:enumeration value='XOR'></xs:enumeration>
      <xs:enumeration value='LOOP-COND'></xs:enumeration>
      <xs:enumeration value='LOOP-ITER'></xs:enumeration>
    </xs:restriction>
  </xs:simpleType>


  <!-- ContainedComponent -->
  <xs:element name='ContainedComponent'>
    <xs:annotation><xs:documentation>
    ContainedComponent element: This element represents
    each of the components involved in a composition.
    A Contained Component can be either an integrated
    component or a web service.
    </xs:documentation></xs:annotation>
    <xs:complexType>
      <xs:extension base='VisualObject'>
        <xs:all>
          <xs:element maxOccurs='1' minOccurs='1'
           ref='ComponentFeatures'></xs:element>
          <xs:element maxOccurs='1' minOccurs='1'
           ref='Inputs'></xs:element>
          <xs:element maxOccurs='1' minOccurs='1'
           ref='Outputs'></xs:element>
```

```
      </xs:all>
    </xs:extension>
  </xs:complexType>
</xs:element>


<!-- Binding -->
<xs:element name='Binding'>
  <xs:annotation><xs:documentation>
  Binding element: A Binding expresses how the data
  flows from a source parameter to a target one.
  </xs:documentation></xs:annotation>
  <xs:complexType>
    <xs:extension base='Object'>
      <xs:attribute name='role' type='BindingRole'
       use='required'></xs:attribute>
      <xs:all>
        <xs:element maxOccurs='1' minOccurs='1'
         ref='CopyFrom'></xs:element>
        <xs:element maxOccurs='1' minOccurs='1'
         ref='CopyTo'></xs:element>
      </xs:all>
    </xs:extension>
  </xs:complexType>
</xs:element>


<xs:simpleType name='BindingRole'>
  <xs:restriction base='xs:string'>
    <xs:enumeration value='input'></xs:enumeration>
    <xs:enumeration value='output'></xs:enumeration>
    <xs:enumeration value='internal'></xs:enumeration>
    <xs:enumeration value='constant'></xs:enumeration>
  </xs:restriction>
</xs:simpleType>


<!-- CopyFrom -->
<xs:element name='CopyFrom'>
  <xs:annotation><xs:documentation>
  CopyFrom element: This element selects the source
  paramenter for data flow.
  </xs:documentation></xs:annotation>
  <xs:complexType>
```

```
      <xs:extension base='tExtensibleBase'>
        <xs:attribute name='idParam' type='xs:ID'
         use='required'></xs:attribute>
        <xs:attribute name='idContained' type='xs:ID'>
         </xs:attribute>
        <xs:attribute name='type' type='BindingType'
         use='required'></xs:attribute>
        <xs:all>
          <xs:element maxOccurs='1' minOccurs='1'
           ref='Name' type='xs:string'>
          </xs:element>
          <xs:element maxOccurs='1' minOccurs='0'
           name='Query' type='xs:string'>
          </xs:element>
        </xs:all>
      </xs:extension>
    </xs:complexType>
  </xs:element>


  <!-- CopyTo -->
  <xs:element name='CopyTo'>
    <xs:annotation><xs:documentation>
    CopyTo element: This element selects the target
    parameter for data flow.
    </xs:documentation></xs:annotation>
    <xs:complexType>
      <xs:extension base='tExtensibleBase'>
      <xs:attribute name='idParam' type='xs:ID'
       use='required'></xs:attribute>
      <xs:attribute name='idContained' type='xs:ID'>
      </xs:attribute>
      <xs:attribute name='type' type='BindingType'
       use='required'></xs:attribute>
      <xs:all>
          <xs:element maxOccurs='1' minOccurs='1'
           ref='Name' type='xs:string'>
          </xs:element>
          <xs:element maxOccurs='1' minOccurs='0'
           name='Query' type='xs:string'>
          </xs:element>
        </xs:all>
      </xs:extension>
    </xs:complexType>
```

```
  </xs:element>


  <xs:simpleType name='BindingType'>
  <xs:restriction base='xs:string'>
    <xs:enumeration value='input'></xs:enumeration>
    <xs:enumeration value='output'></xs:enumeration>
  </xs:restriction>
  </xs:simpleType>

</xs:schema>
```

ANEXO B

Proceso WSBPEL para el caso de aplicación

En este anexo se incluye el proceso WSBPEL generado para el componente integrado `info viento` en el escenario de control de emergencias presentado en el capítulo 5. No hemos incluido todo el proceso para el componente integrado `monitorización` por la extensión del código WSBPEL resultante.

Proceso WSBPEL

```
<?xml version='1.0' encoding='UTF-8'?>

<!--

getWindInfoPlace BPEL Process generated by the Integrated
Component Designer

-->

<process
 xmlns='http://schemas.xmlsoap.org/ws/2003/03/business-process/'
 xmlns:tns='http://www.geoinfo.uji.es/process'
 xmlns:xsd='http://www.w3.org/2001/XMLSchema'
 xmlns:bpws='http://schemas.xmlsoap.org/ws/2003/03/business-process/'
```

```
 xmlns:bpelx='http://schemas.oracle.com/bpel/extension'
 xmlns:ora='http://schemas.oracle.com/xpath/extension'
 xmlns:nmxml0='http://examples.org/wsdl/ICGazetteer.wsdl'
 xmlns:nmxml1='http://localhost:8080/axis/services/ADLGazClient'
 xmlns:nmxml2='http://examples.org/wsdl/ICProximity.wsdl'
 xmlns:nmxml3='http://www.ionicsoft.com/acegis'
 xmlns:nmxml4='http://examples.org/wsdl/ICMeteo.wsdl'
 xmlns:nmxml5='http://www.capeclear.com/AirportWeather.wsdl'
 targetNamespace='http://www.geoinfo.uji.es/process'
 name='getWindInfoPlace'
 suppressJoinFailure='yes'>

<!--

PARTNERLINKS
List of services participating in this BPEL process
The 'clientPL' role represents the requester of this service

-->

  <partnerLinks>
    <partnerLink name='clientPL'
      partnerLinkType='tns:processPLType'
      myRole='processProvider'/>
    <partnerLink name='AirportWeatherPL'
      partnerLinkType='nmxml5:AirportWeatherPLType'
      partnerRole='AirportWeatherProvider'/>
    <partnerLink name='ADLGazClientPL'
      partnerLinkType='nmxml1:ADLGazClientPLType'
      partnerRole='ADLGazClientProvider'/>
    <partnerLink name='GetNearestAirportCodePL'
      partnerLinkType='nmxml3:GetNearestAirportCodePLType'
      partnerRole='GetNearestAirportCodeProvider'/>
  </partnerLinks>

<!--

VARIABLES
List of messages and XML documents used within this BPEL process

-->

  <variables>
    <variable name='input'
```

```
      messageType='tns:processReqMsg'/>
    <variable name='getWindInfoInput'
      messageType='nmxml4:ReqMsg'/>
    <variable name='getWindInput'
      messageType='nmxml5:getWindRequestMessage'/>
    <variable name='getADLCoordinatesInput'
      messageType='nmxml1:getCoordinatesRequest'/>
    <variable name='getCoordinatesInput'
      messageType='nmxml0:ReqMsg'/>
    <variable name='getNearestAirportInput'
      messageType='nmxml2:ReqMsg'/>
    <variable name='getNearestAirportCodeInput'
      messageType='nmxml3:getNearestAirportCodeRequestMessage'/>
    <variable name='output'
      messageType='tns:processResMsg'/>
    <variable name='getWindInfoOutput'
      messageType='nmxml4:ResMsg'/>
    <variable name='getWindOutput'
      messageType='nmxml5:getWindResponseMessage'/>
    <variable name='getADLCoordinatesOutput'
      messageType='nmxml1:getCoordinatesResponse'/>
    <variable name='getCoordinatesOutput'
      messageType='nmxml0:ResMsg'/>
    <variable name='getNearestAirportOutput'
      messageType='nmxml2:ResMsg'/>
    <variable name='getNearestAirportCodeOutput'
      messageType='nmxml3:getNearestAirportCodeResponseMessage'/>
  </variables>

<!--
ORCHESTRATION LOGIC
The following represents the selection and composition
patterns transformed into BPEL code.

Receive input from requester by the
receive activity 'receiveInput'.
This maps to operation defined in getWindInfoAtPlace.wsdl

Generate reply to synchronous request by the
reply activity 'replyOutput'.
This maps to operation defined in getWindInfoAtPlace.wsdl

-->
```

```
<sequence name='main'>
  <receive name='receiveInput'
    partnerLink='clientPL'
    portType='tns:processPT'
    operation='process' variable='input'
    createInstance='yes'/>
  <scope name='SEQ'>
    <sequence>
      <assign name='input'>
        <copy>
          <from variable='input'
            part='payload'
            query='/tns:inputType'/>
          <to variable='getCoordinatesInput'
            part='name'/>
        </copy>
      </assign>
      <scope name='AND-DISC'
        variableAccessSerializable='no'>
        <sequence>
          <variables>
            <variable name='failService0'
              type='xsd:boolean'>true()
            </variable>
          </variables>
          <faultHandlers>
            <catchAll>
              <empty/>
            </catchAll>
          </faultHandlers>
          <sequence>
            <assign name='input'>
              <copy>
                <from variable='getCoordinatesInput'
                  part='name'/>
                <to variable='getADLCoordinatesInput'
                  part='name'/>
              </copy>
            </assign>
            <invoke name='getADLCoordinates'
              partnerLink='getADLCoordinates'
              portType='nmxml1:ADLGazClient'
              operation='getADLCoordinates'
              inputVariable='getADLCoordinatesInput'
```

```
        outputVariable='getADLCoordinatesOutput'>
        <compensationHandler>
          <assign>
            <copy>
              <from expression='false()'/>
              <to variable='failService0'/>
            </copy>
            <copy>
              <from variable='getADLCoordinatesOutput'
                part='output'/>
              <to variable='getCoordinatesOutput'
                part='coordinates'/>
            </copy>
          </assign>
        </compensationHandler>
      </invoke>
    </sequence>
  </sequence>
</scope>
<assign name='internal'>
  <copy>
    <from variable='getCoordinatesOutput'
      part='coordinates'/>
    <to variable='getNearestAirportInput'
      part='point'/>
  </copy>
</assign>
<scope name='AND-DISC'
  variableAccessSerializable='no'>
  <sequence>
    <variables>
      <variable name='failService0'
        type='xsd:boolean'>true()
      </variable>
    </variables>
    <faultHandlers>
      <catchAll>
        <empty/>
      </catchAll>
    </faultHandlers>
    <sequence>
      <assign name='input'>
        <copy>
          <from variable='getNearestAirportInput'
```

```
                part='point'/>
              <to variable='getNearestAirportCodeInput'
                part='point'/>
            </copy>
          </assign>
          <invoke name='getNearestAirportCode'
            partnerLink='getNearestAirportCode'
            portType='nmxml3:GetNearestAirportCodePT'
            operation='getNearestAirportCode'
            inputVariable='getNearestAirportCodeInput'
            outputVariable='getNearestAirportCodeOutput'>
            <compensationHandler>
              <assign>
                <copy>
                  <from expression='false()'/>
                  <to variable='failService0'/>
                </copy>
                <copy>
                  <from variable='getNearestAirportCodeOutput'
                    part='AirportCodeReturn'/>
                  <to variable='getNearestAirportOutput'
                    part='code'/>
                </copy>
              </assign>
            </compensationHandler>
          </invoke>
        </sequence>
      </sequence>
    </scope>
    <assign name='internal'>
      <copy>
        <from variable='getNearestAirportOutput'
          part='code'/>
        <to variable='getWindInfoInput'
          part='code'/>
      </copy>
    </assign>
    <scope name='AND-DISC'
      variableAccessSerializable='no'>
      <sequence>
        <variables>
          <variable name='failService0'
            type='xsd:boolean'>true()
          </variable>
```

```
    </variables>
    <faultHandlers>
      <catchAll>
        <empty/>
      </catchAll>
    </faultHandlers>
    <sequence>
      <assign name='input'>
        <copy>
          <from variable='getWindInfoInput'
            part='code'/>
          <to variable='getWindInput'
            part='arg0'/>
        </copy>
      </assign>
      <invoke name='getWind'
        partnerLink='getWind'
        portType='nmxml5:AirportWeatherPT'
        operation='getWind'
        inputVariable='getWindInput'
        outputVariable='getWindOutput'>
        <compensationHandler>
          <assign>
            <copy>
              <from expression='false()'/>
              <to variable='failService0'/>
            </copy>
            <copy>
              <from variable='getWindOutput'
                part='return'/>
              <to variable='getWindInfoOutput'
                part='windInfo'/>
            </copy>
          </assign>
        </compensationHandler>
      </invoke>
    </sequence>
  </sequence>
</scope>
<assign name='output'>
  <copy>
    <from variable='getWindInfoOutput'
      part='windInfo'/>
    <to variable='output'
```

```
              part='payload'
              query='/tns:outputType'/>
          </copy>
        </assign>
      </sequence>
    </scope>
    <reply name='replyOutput'
      partnerLink='clientPL'
      portType='tns:processPT'
      operation='process'
      variable='output'/>
  </sequence>
</process>
```

Descripción WSDL del proceso

```
<?xml version='1.0' encoding='UTF-8'?>

<!--

WSDL file generated automatically for
the process getWindInfoAtPlace.bpel

-->

<definitions
  xmlns='http://schemas.xmlsoap.org/wsdl/'
  xmlns:xsd='http://www.w3.org/2001/XMLSchema'
  xmlns:plnk='http://schemas.xmlsoap.org/ws/2003/05/partner-link/'
  xmlns:nmxml0='http://examples.org/wsdl/ICGazetteer.wsdl'
  xmlns:nmxml1='http://localhost:8080/axis/services/ADLGazClient'
  xmlns:nmxml2='http://examples.org/wsdl/ICProximity.wsdl'
  xmlns:nmxml3='http://www.ionicsoft.com/acegis'
  xmlns:nmxml4='http://examples.org/wsdl/ICMeteo.wsdl'
  xmlns:nmxml5='http://www.capeclear.com/AirportWeather.wsdl'
  xmlns:tns='http://www.geoinfo.uji.es/process'
  targetNamespace='http://www.geoinfo.uji.es/process'
  name='getWindInfoAtPlace'>

<!--
```

```
TYPES DEFINITIONS
List of types participating in
this BPEL process

-->

  <types>
    <schema xmlns='http://www.w3.org/2001/XMLSchema'
      targetNamespace='http://www.geoinfo.uji.es/process'>
      <complexType name='anyType'>
        <sequence>
          <any minOccurs='0' maxOccurs='unbounded'/>
        </sequence>
      </complexType>
      <element name='inputType' type='tns:anyType'/>
      <element name='outputType' type='tns:anyType'/>
    </schema>
  </types>

<!--

MESSAGE TYPE DEFINITIONS
Definition of the message types used as part
of the port types definitions

-->

  <message name='processReqMsg'>
    <part name='payload' element='tns:inputType'/>
  </message>
  <message name='processResMsg'>
    <part name='payload' element='tns:outputType'/>
  </message>

<!--

PORT TYPE DEFINITIONS
A port type groups a set of operations into
a logical service unit

-->

  <portType name='processPT'>
    <operation name='process'>
```

```
      <input message='tns:processReqMsg'/>
      <output message='tns:processResMsg'/>
    </operation>
  </portType>

<!--

PARTNER LINK TYPE DEFINITIONS
List of service roles participating within this process

-->

  <plnk:partnerLinkType name='processPLType'>
    <plnk:role name='processProvider'>
      <plnk:portType name='tns:processPT'/>
    </plnk:role>
  </plnk:partnerLinkType>
  <plnk:partnerLinkType name='AirportWeatherPLType'>
    <plnk:role name='AirportWeatherProvider'>
      <plnk:portType name='nmxml5:AirportWeatherPT'/>
    </plnk:role>
  </plnk:partnerLinkType>
  <plnk:partnerLinkType name='ADLGazClientPLType'>
    <plnk:role name='ADLGazClientProvider'>
      <plnk:portType name='nmxml1:ADLGazClient'/>
    </plnk:role>
  </plnk:partnerLinkType>
  <plnk:partnerLinkType name='GetNearestAirportCodePLType'>
    <plnk:role name='GetNearestAirportCodeProvider'>
      <plnk:portType name='nmxml3:GetNearestAirportCodePT'/>
    </plnk:role>
  </plnk:partnerLinkType>
</definitions>
```

Bibliografía

[1] Survey: E-commerce. *The Economist 371*, 8375 (2004), 53–69.

[2] Abel, D. J. Towards integrated geographical information processing. *International Journal of Geographical Information Science 12*, 4 (1998), 353–371.

[3] Abiteboul, S., Buneman, P., and Suciu, D. *Data on the Web*. Morgan Kaufmann, San Francisco, 2000.

[4] Agarwal, S., Handschuh, S., and Staab, S. Annotation, composition and invocation of semantic web services. *Journal of Web Semantics 2*, 1 (2004), 31–48.

[5] Aissi, S., Malu, P., and Srinivasan, K. E-business process modeling: The next big step. *IEEE Computer Magazine 35*, 5 (2002), 55–62.

[6] Alameh, N. Chaining geographic information web services. *IEEE Internet Computing 7*, 5 (2003), 22–29.

[7] Alonso, G., Casati, F., Kuno, H., and Machiraju, V. *Web Services. Concepts, Architectures and Applications*. Springer, Berlín, 2004.

[8] An, Y., Zhao, B., and Bian, F. Geo web services based on semantic. In *Conceptual Modeling for Advanced Application Domains: ER 2004 Workshops* (Shanghai, China, 2004), Springer, Berlín, 2004, pp. 139–147.

[9] Anzböck, R., and Dustdar, S. Modeling and implementing medical web services. *Data & Knowledge Engineering 55*, 2 (2005), 203–236.

[10] Anzböck, R., and Dustdar, S. Semi-automatic generation of web services and BPEL processes - a model-driven approach. In *Business Process Management: 3rd International Conference* (Nancy, France, 2005), Springer, Berlín, 2005, pp. 64–79.

[11] Bañares, J., Bernabé, M. A., Gould, M., Muro-Medrano, P. R., and Nogueras, J. Infraestructura nacional de información geográfica y su utilidad para las administraciones públicas. *Boletic*, Septiembre-Octubre (2001), 51–60.

[12] Baïna, K., Benatallah, B., Casati, F., and Toumani, F. Model-driven web service development. In *Advanced Information Systems Engineering: 16th International Conference* (Riga, Latvia, 2004), Springer, Berlín, 2004, pp. 290–306.

[13] Bansal, A., Patel, K., Gupta, G., Raghavachari, B., Harris, E., and Staves, J. C. Towards intelligent services: A case study in chemical emergency response. In *IEEE International Conference on Web Services* (Orlando, USA, 2005), IEEE CS Press, Los Alamitos, 2005, pp. 751–758.

[14] Bassiliades, N., Anagnostopopoulos, D., and Vlahavas, I. Web service composition using a deductive XML rule language. *Distributed and Parallel Databases 17*, 2 (2005), 135–178.

[15] Benatallah, B., Casati, F., and Toumani, F. Web service conversation modeling. *IEEE Internet Computing 8*, 1 (2004), 46–54.

[16] Benatallah, B., Dumas, M., Fauvet, M.-C., and Rabhi, F. Towards patterns of web services composition. In *Patterns and skeletons for parallel and distributed computing*, F. Rabhi and S. Gorlatch, Eds. Springer, London, 2003, pp. 265–296.

[17] Benatallah, B., Dumas, M., Fauvet, M.-C., Rabhi, F., and Sheng, Q. Z. Overview of some patterns for architecting and managing composite web services. *ACM SIGecom Exchanges 3*, 3 (2002), 9–16.

[18] Benatallah, B., Dumas, M., and Maamar, Z. Definition and execution of composite web services: The self-serv project. *IEEE Data Engineering 25*, 4 (2002), 47–52.

[19] Benatallah, B., Dumas, M., and Sheng, Q. Z. Facilitating the rapid development and scalable orchestration of composite web services. *Distributed and Parallel Databases 17*, 1 (2005), 5–37.

[20] Benatallah, B., Sheng, Q. Z., and Dumas, M. The self-serv environment for web services composition. *IEEE Internet Computing 7*, 1 (2003), 40–48.

[21] Bernard, L., Craglia, M., Gould, M., and Kuhn, W. Towards an SDI research agenda. In *Abstract Handbook of the 11th EC-GIS Workshop* (Sardinia, Italy, 2005), European Commission Joint Research Centre, Ispra, 2005, pp. 147–151.

[22] Bernard, L., Einspanier, U., Lutz, M., and Portele, C. Interoperability in GI service chains - the way forward. In *Proceedings of the 6th AGILE International Conference on Geographic Information Science* (Lyon, France, 2003), Presses Polytechniques et Universitaires Romandes, Lausanne, 2003, pp. 179–187.

[23] Berners-Lee, T., Hendler, J., and Lassila, O. The semantic web. *Scientific American 284*, 5 (2001), 34–43.

[24] Biggerstaff, T. J., and Perlis, A. J. *Software reusability: vol. 1, concepts and models*. ACM Press, New York, 1989.

[25] Bishr, Y. Overcoming the semantic and other barrier to GIS interoperability. *International Journal of Geographical Information Science 12*, 4 (1998), 299–313.

[26] Bodoff, D., Hung, P. C., and Ben-Menachem, M. Web metadata standards: Observations and prescriptions. *IEEE Software 20*, 1 (2005), 78–85.

[27] Bussler, C. The application of workflow technology in semantic B2B integration. *Distributed and Parallel Databases 12*, 2-3 (2002), 163–191.

[28] Bussler, C. Semantic web services: Reflections on web service mediation and composition. In *Proceedings of the 4th International Conference on Web Information Systems Engineering* (Rome, Italy, 2003), IEEE CS Press, Los Alamitos, 2003, pp. 253–260.

[29] Bussler, C., Fensel, D., and Maedche, A. A conceptual architecture for semantic web enabled web services. *ACM SIGMOD Record 31*, 4 (2002), 24–29.

[30] Cabral, L., Domingue, J., Motta, E., Payne, T., and Hakimpour, F. Approaches to semantic web services: An overview and comparisons. In *The Semantic Web: Research and Applications: First European Semantic Web Symposium* (Heraklion, Greece, 2004), Springer, Berlín, 2004, pp. 225–239.

[31] Cardoso, J., and Sheth, A. P. Semantic e-workflow composition. *Journal of Intelligent Information Systems 21*, 3 (2003), 191–225.

[32] Cardoso, J., Sheth, A. P., Miller, J. A., Arnold, J., and Kochut, K. Quality of service for workflows and web service processes. *Journal of Web Semantics 1*, 3 (2004), 281–308.

[33] Casati, F., Eric, S., Dayal, U., and Shan, M.-C. Business-oriented managament of web services. *Communications of the ACM 46*, 10 (2003), 55–60.

[34] Casati, F., and Shan, M.-C. Dynamic and adaptative composition of e-services. *Information Systems 26*, 3 (2001), 143–163.

[35] Chen, L., Shadbolt, N., Goble, C., Tao, F., Cox, S., Puleston, C., and Smart, P. Towards a knowledge-based approach to semantic service composition. In *The SemanticWeb - ISWC 2003* (Sanibel Island, USA, 2003), Springer Berlín, 2003, pp. 319–334.

[36] Clayberg, E., and Rubel, D. *Eclipse: Building Commercial-Quality Plug-ins*. Pearson Education, Boston, 2004.

[37] Corporation, M. Dcom technical overview, December 1996. `http://msdn.microsoft.com/library/` (Accessed Sep 2005).

[38] Cost, R. S., Finin, T., Joshi, A., Peng, Y., Nicholas, C., Soboroff, I., Chen, H., Kagal, L., Perich, F., Zou, Y., and Tolia, S. Ittalks: A case study in the semantic web and DAML+OIL. *IEEE Intelligent Systems 17*, 1 (2002), 40–47.

[39] Crnkovic, I. Component-based software engineering – new challenges in software development. *Journal of Computing and Information Technology 11*, 3 (2003), 151–161.

[40] Curbera, F. Web services: Standards based distributed computing. *UPGRADE, The European Journal for the Informatics Professional 6*, 1 (2005), 45–50.

[41] Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., and Weerawarana, S. Unraveling the web services: An introduction to SOAP, WSDL, and UDDI. *IEEE Internet Computing 6*, 2 (2002), 86–93.

[42] Cutter, S. L. GI science, disasters, and emergency management. *Transactions in GIS 7*, 4 (2003), 436–445.

[43] Davies, N. J., Fensel, D., and Richardson, M. The future of web services. *BT Technology Journal 22*, 1 (2004), 118–130.

[44] de Bruijn, J., Fensel, D., Keller, U., and Lara, R. Using the web service modeling ontology to enable semantic e-business. *Communications of the ACM 48*, 12 (2005), 43–47.

[45] de Bruijn, J., Lara, R., Arroyo, S., Gómez, J. M., Han, S.-K., and Fensel, D. A unified semantic web services architecture based on WSMF and UPML. *International Journal of Web Engineering and Technology 2*, 2-3 (2004), 148–180.

[46] Dustdar, S., and Schreiner, W. A survey on web services composition. *International Journal of Grid and Web Services 1*, 1 (2005), 1–30.

[47] Fensel, D., and Bussler, C. The web service modeling framework WSMF. *Electronic Commerce Research and Applications 1*, 2 (2002), 113–137.

[48] Fielding, R. T. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[49] Fonseca, F., Davis, C., and Câmara, G. Brinding ontologies and conceptual schemas in geographic information integration. *GeoInformatica 7*, 4 (2003), 355–378.

[50] Fonseca, F., Egenhofer, M., Agouris, P., and Câmara, G. Using ontologies for integrated geographic information systems. *Transactions in GIS 6*, 3 (2002), 231–257.

[51] Geer, D. Eclipse becomes the dominant java IDE. *IEEE Computer Magazine 38*, 7 (2005), 16–18.

[52] Gómez-Pérez, A., and Corcho, s. Ontology languages for the semantic web. *IEEE Intelligent Systems 17*, 1 (2002), 54–60.

[53] Gómez-Pérez, A., González-Cabero, R., and Lama, M. ODE SWS: A framework for designning and composing semantic web services. *IEEE Intelligent Systems 19*, 4 (2004), 24–31.

[54] Granell, C., and Gould, M. An integrated component-based framework for service reuse. *Submitted to International Journal of Information Technology and Web Engineering* (2006).

[55] Granell, C., Gould, M., Grønmo, R., and Skogan, D. Improving reuse of web service compositions. In *International Conference on E-Commerce and Web Technologies* (Copenhagen, Denmark, 2005), Springer, Berlín, 2005, pp. 358–367.

[56] Granell, C., Gould, M., Lemmens, R., Wytzisk, A., de By, R., and van Oosterom, P. Integrating semantic and syntactic descriptions for chaining geographic services. *Submitted to IEEE Internet Computing* (2006).

[57] Granell, C., Gould, M., and Ramos, J. F. Service composition for SDIs: Integrated components creation. In *16th International Workshop on Database and Expert Systems Applications* (Copenhagen, Denmark, 2005), IEEE CS Press, Los Alamitos, 2005, pp. 475–479.

[58] Granell, C., Poveda, J., and Gould, M. Incremental weak composition and invocation of geographic web services. In *International Workshop on Semantic Processing of Spatial Data* (Mexico D.F., Mexico, 2003), Instituto Politécnico Nacional, Mexico, 2003, pp. 179–187.

[59] Granell, C., Poveda, J., and Gould, M. An incremental approach to web service composition. *Colombian Journal of Computation 5*, 1 (2004), 22–34.

[60] Granell, C., Poveda, J., and Gould, M. Incremental composition of geograhic web services: An emergency management context. In *7th AGILE International Conference on Geographic Information Science* (Heraklion, 2004), Crete University Press, Greece, 2004, pp. 343–348.

[61] Granell, C., and Ramos, J. F. An object-oriented approach to GI web service composition. In *15th International Workshop on Database and Expert Systems Applications* (Zaragoza, Spain, 2004), IEEE CS Press, Los Alamitos, 2004, pp. 835–839.

[62] Grønmo, R., Skogan, D., Solheim, I., and Oldevick, J. Model-driven web service development. *International Journal on Web Services Research 1*, 4 (2004), 1–13.

[63] Hamadi, R., and Benatallah, B. A petri net-based model for web service composition. In *Proceedings of the 14th Australasian database conference on Database technologies* (Adelaide, Australia, 2003), Australian Computer Society, Darlinghurst, 2003, pp. 191–200.

[64] Hart, G., and Greenwood, J. A component based approach to geo-ontologies and geodata modelling to enable data sharing. In *Proceedings of the 6th AGILE International Conference on Geographic Information Science* (Lyon, France, 2003), Presses Polytechniques et Universitaires Romandes, Laussane, 2003, pp. 198–206.

[65] Havey, M. *Essential Business Process Modeling.* O'Reilly, Sebastopol, 2005.

[66] Hayes, J. G., Peyrovian, E., Sarin, S., Schimidt, M., Swenson, K. D., and Weber, R. Workflow interoperability standards for the internet. *IEEE Internet Computing 4*, 3 (2000), 37–45.

[67] Henderson, P., and Yang, J. Reusable web services. In *Software Reuse: Methods, Techniques and Tools: 8th International Conference* (Madrid, Spain, 2004), Springer, Berlín, 2004, pp. 185–194.

[68] Hendler, J. Agents and the semantic web. *IEEE Intelligent Systems 16*, 2 (2001), 30–37.

[69] Huhns, M. N., and Singh, M. P. Service-oriented computing: Key concepts and principles. *IEEE Internet Computing 9*, 1 (2005), 75–81.

[70] ISO 19119:2005. Geographic information – services.

[71] Jaeger, M. C., Rojec-Goldmann, G., and Mühl, G. QoS aggregation for web service composition using workflow patterns. In *Enterprise Distributed Object Computing, 8th IEEE International Conference* (Monterrey, USA, 2004), IEEE CS Press, Los Alamitos, 2004, pp. 149–151.

[72] Kawamura, T., De Blasio, J.-A., Hasegawa, T., Paolucci, M., and Sycara, K. Public deployment of semantic service matchmaker with UDDI bussines registry. In *The Semantic Web - ISWC 2004: Third International Semantic Web Conference* (Hiroshima, Japan), Springer, Berlín, 2004, pp. 752–766.

[73] Kawamura, T., De Blasio, J.-A., Hasegawa, T., Paolucci, M., and Sycara, K. Preliminary report of public experiment of semantic service matchmaker with UDDI business registry. In *Service-Oriented Computing - ICSOC 2003* (Trento, Italy, 2003), Springer, Berlín, 2003, pp. 208–224.

[74] Kaye, D. *Loosely coupled: the missing pieces of Web services.* RDS Press, Marin County, 2003.

[75] Khalaf, R., Mukhi, N., and Weerawarana, S. Service-oriented composition in BPEL4WS [on-line]. In *Proceedings of the 12th International World Wide Web Conference - Alternate Paper Tracks* (Budapest, Hungary, 2003). `http://www2003.org/cdrom/papers/alternate/P768/choreo_html/p768-khalaf.htm` (Accessed May 2006).

[76] Kiepuszewski, B. *Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows.* PhD thesis, Queensland University of Technology, 2003.

[77] Kim, M., Kim, M., Lee, E., and Joo, I. Web services framework for geo-spatial services. In *Web and Wireless Geographical Information Systems: 4th International Workshop* (Goyand, Korea, 2004), Springer, Berlín, 2005, pp. 1–13.

[78] Klien, E., Lutz, M., and Kuhn, W. Ontology-based discovery of geographic information services–an application in disaster management. *Computers, Enviroment and Urban Systems 30*, 1 (2006), 102–123.

[79] Ko, I.-Y., and Neches, R. Composing web services for large-scale tasks. *IEEE Internet Computing 7*, 5 (2003), 52–59.

[80] Krasner, G. E., and Pope, S. T. A cookbook for using the model-view-controller user interface paradigm in smalltalk-80. *Journal of Object Oriented Programming 1*, 3 (1988), 26–49.

[81] Lara, R., Lausen, H., Arroyo, S., de Bruijn, J., and Fensel, D. Semantic web services: description requirements and current technologies [on-line]. In *International Workshop on Electronic Commerce, Agents, and Semantic Web Services* (Pittsburgh, USA, 2003). `http://www.debruijn.net/publications/sws-description.pdf` (Accessed May 2006).

[82] Lara, R., Roman, D., Polleres, A., and Fensel, D. A conceptual comparison of WSMO and OWL-S. In *Web Services: European Conference* (Erfurt, Germany, 2004), Springer, Berlín, 2004, pp. 254–269.

[83] Laukkanen, M., and Helin, H. Composing workflows of semantic web services [on-line]. In *Workshop on Web Services and Agent-based Engineering* (Melbourne, Australia, 2003). `http://www.agentus.com/WSABE2003/program/laukkanen.pdf` (Accessed May 2006).

[84] Lee, E., Kim, M., Kim, M., and Joo, I. A web services framework for integrated geospatial coverage data. In *Computational Science and Its Applications - ICCSA 2005 Part II: International Conference* (Singapure, 2005), Springer, Berlín, 2005, pp. 1136–1145.

[85] Lemmens, R. *Semantic Interoperability of Distributed Geo-Services*. PhD thesis, International Institute for Geo-Information Science and Earth Observation, 2006.

[86] Lemmens, R., Granell, C., Wytzisk, A., de By, R., Gould, M., and van Oosterom, P. Semantic and syntactic service descriptions at work in geo-service chaining. In *Proceedings of the 9th AGILE International Conference on Geographic Information Science* (Visegrád, Hungary, 2006), University of West Hungary, Székesfehérvár, 2006, pp. 51–61.

[87] Li, S., and Coleman, D. Modeling distributed GIS data production workflow. *Computers, Enviroment and Urban Systems 29*, 4 (2005), 401–424.

[88] Maguire, D. J., and Longley, P. A. The emergence of geoportals and their role in spatial data infrastructures. *Computers, Environment and Urban Systems 29*, 1 (2005), 3–4.

[89] Mandell, D. J., and McIlraith, S. A. Adapting BPEL4WS for the semantic web: The bottom-up approach to web service interoperation. In *The SemanticWeb - ISWC 2003* (Sanibel Island, USA, 2003), Springer, Berlín, 2003, pp. 227–241.

[90] McGuinnes, D. L., Fikes, R., Hendler, J., and Stein, L. A. DAML+OIL: An ontology language for the semantic web. *IEEE Intelligent Systems 17*, 5 (2002), 72–80.

[91] McIlraith, S. A., Son, T. C., and Zeng, H. Semantic web services. *IEEE Intelligent Systems 16*, 2 (2001), 46–53.

[92] McIlroy, M. D. Mass-produced software component. In *Proceedings of the Working Conference on Software Engineering* (Garmisch-Partenkirchen, Germany, 1968), NATO Science Committee, Brussels, 1969, pp. 138–150.

[93] Medeiros, C., Perez-Alcazar, J., Digiampietri, L., Pastorello, Jr, G. Z., Santache, A., Torres, R., Madeira, E., and Bacarin, E. WOODS and the web: annotating and reusing scientific workflows. *ACM SIGMOD Record 34*, 3 (2005), 18–23.

[94] Medjahed, B., Benatallah, B., Bouguettaya, A., Ngu, A., and Elmagarmid, A. Business-to-business interactions: issues and enabling technologies. *The International Journal on Very Large Data Bases 12*, 1 (2003), 59–85.

[95] Medjahed, B., Bouguettaya, A., and Elmagarmid, A. Composing web services on the semantic web. *The International Journal on Very Large Data Bases 12*, 4 (2003), 333–351.

[96] Melloul, L., and Fox, A. Reusable functional composition patterns for web services. In *International Conference on Web Services* (San Diego, USA, 2004), IEEE CS Press, Los Alamitos, 2004, pp. 498–505.

[97] Menascé, D. A. QoS issues in web services. *IEEE Internet Computing 6*, 6 (2002), 72–75.

[98] Menascé, D. A. Composing web services: A QoS view. *IEEE Internet Computing 8*, 6 (2004), 88–90.

[99] Meredith, L., and Bjorg, S. Contracts and types. *Communications of the ACM 46*, 10 (2003), 41–47.

[100] Milanovic, N., and Malek, M. Current solutions for web service composition. *IEEE Internet Computing 8*, 6 (2004), 51–59.

[101] Milner, R. *Communicating and Mobile Systems: The Pi Calculus*. Cambridge Universisty Press, Cambridge, 1999.

[102] Mitra, P., and Wiederhold, G. Resolving terminological heterogeneity in ontologies. In *Proceedings of the ECAI-02 Workshop on Ontologies and Semantic Interoperability* (Lyon, France, 2002), 2002, pp. 45–50.

[103] Narayanan, S., and McIlraith, S. A. Simulation, verification and automated composition of web services. In *Proceedings of the 11th international conference on World Wide Web* (Honolulu, USA, 2002), ACM Press, New York, 2002, pp. 77–88.

[104] Nebert, D. Developing spatial data infrastructures: The SDI cookbook version 2.0. Tech. rep., Global Spatial Data Infrastructure (GSDI), 2004.

[105] OASIS. Oasis web services business process exection language version 2.0, 21 December 2005. `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel` (Accessed May 2006).

[106] OGC. OWS1 coverage portrayal service. Tech. rep., Open Geospatial Consortium, 2002.

[107] OGC. Styled layer description implementation specification. Tech. rep., Open Geospatial Consortium, 2002.

[108] OGC. Web coverage service. Tech. rep., Open Geospatial Consortium, 2003.

[109] OGC. OpenGIS geography markup language (GML) encoding specification. Tech. rep., Open Geospatial Consortium, 2004.

[110] OGC. OGC catalogue services specification. Tech. rep., Open Geospatial Consortium, 2005.

[111] OGC. Opengis web processing service. Tech. rep., Open Geospatial Consortium, 2005.

[112] OGC. Web feature service implementation specification. Tech. rep., Open Geospatial Consortium, 2005.

[113] OGC. Web map service implementation specification. Tech. rep., Open Geospatial Consortium, 2005.

[114] Onchanga, R. Quality of service management framework for dynamic chaining of geographic information services. *International Journal of Applied Earth Observation and Geoinformation 8*, 2 (2006), 137–148.

[115] Ort, E. Ease of development of entreprise javebeans technology, October 2004. `http://java.sun.com/developer/technicalArticles/ebeans/ejbease/index.html` (Accessed Apr 2006).

[116] O'Sullivan, J., Edmond, D., and ter Hofstede, A. H. What's in a service?: Towards accurate description of non-functional service properties. *Distributed and Parallel Databases 12*, 2-3 (2002), 117–133.

[117] Pahl, C. A conceptual framework for semantic web services development and deployment. In *Web Services: European Conference* (Erfurt, Germany, 2004), Springer, Berlín, 2004, pp. 270–284.

[118] Paolucci, M., and Sycara, K. Autonomous semantic web services. *IEEE Internet Computing 7*, 5 (2003), 34–41.

[119] Pasley, J. How BPEL and SOA are changing web services development. *IEEE Internet Computing 9*, 3 (2005), 60–67.

[120] Patil, S., and Newcomer, E. ebXML and web services. *IEEE Internet Computing 7*, 3 (2003), 74–82.

[121] Pautasso, C. *A Flexible System for Visual Service Composition*. PhD thesis, Swiss Federal Institute of Technology, 2004.

[122] Pautasso, C., and Alonso, G. Flexible binding for reusable composition of web services. In *Software Composition: 4th International Workshop* (Edinburgh, UK, 2005), Springer, Berlín, 2005, pp. 151–166.

[123] Peer, J., and Vukovic, M. A proposal for a sematic web service description format. In *Web Services: European Conference* (Erfurt, Germany, 2004), Springer, Berlín, 2004, pp. 270–284.

[124] Peltz, C. Web services orchestration and choreography. *IEEE Computer Magazine 36*, 10 (2003), 46–52.

[125] Pistore, M., Traverso, P., Bertoli, P., and Marconi, A. Automated synthesis of composite BPEL4WS web services. In *IEEE International Conference on Web Services* (Orlando, USA, 2005), IEEE CS Press, Los Alamitos, 2005, pp. 293–301.

[126] Poveda, J., Gould, M., and Granell, C. Composition of e-commerce and geographic information services for emergency management. In *Electronic Government: 3rd International Conference* (Zaragoza, Spain, 2004), Springer, Berlín, 2004, pp. 562–563.

[127] Pundt, H., and Bishr, Y. Domain ontologies for data sharing - an example from environmental monitoring using field gis. *Computers & Geosciences 28*, 1 (2002), 95–102.

[128] Rajabifard, A., Feeney, M. F., and Williamson, I. P. Future directions for sdi development. *International Journal of Applied Earth Observation and Geoinformation 4*, 1 (2002), 11–22.

[129] Rao, J., and Su, X. A survey of automated web service composition methods. In *Semantic Web Services and Web Process Composition: First International Workshop* (San Diego, USA, 2004), Springer, Berlín, 2005, pp. 43–54.

[130] Rocha, A., Cestnik, B., and Oliveira, M. A. Interoperable geographic information services to support crisis management. In *Web and Wireless Geographical Information Systems: 5th International Workshop* (Laussane, Switzerland, 2005), Springer, Berlín, 2005, pp. 246–255.

[131] Rosenberg, F., and Dustdar, S. Towards a distributed service-oriented business rules system. In *Proceedings ot the 3rd European Conference on Web Services* (Växjö, Sweden, 2005), IEEE CS Press, Los Alamitos, 2005, pp. 14–24.

[132] Seligman, L., and Rosenthal, A. XML's impact on databases and data sharing. *IEEE Computer Magazine 34*, 6 (2001), 59–67.

[133] Sheth, A. P. Changing focus on interoperability in information systems: From system, syntax, structure to semantics. In *Interoperating Geographic Information Systems*. Kluwer Academic, Dordrecht, 1999, pp. 5–30.

[134] Sheth, A. P., Bertram, C., Avant, D., Hammond, B., Kochut, K., and Warke, Y. Managing semantic content for the web. *IEEE Internet Computing 6*, 4 (2002), 80–87.

[135] Singh, M. P. Physics of service composition. *IEEE Internet Computing 5*, 3 (2001), 6–7.

[136] Sirin, E., Hendler, J., and Parsia, B. Semi-automatic composition of web services using semantic descriptions. In *Web Services: Modeling, Architecture and Infrastructure, Proceedings of the 1st Workshop on Web Services: Modeling, Architecture and Infrastructure* (Angers, France, 2003), ICEIS Press, 2003, pp. 17–24.

[137] Sirin, E., Parsia, B., and Hendler, J. Filtering and selecting semantic web services with interactive composition techniques. *IEEE Intelligent Systems 19*, 4 (2004), 42–49.

[138] Sivashanmugam, K., Verma, K., Sheth, A. P., and Miller, J. A. Adding semantics to web services standards. In *Proceedings of the International Conference on Web Services* (Las Vegas, USA, 2003), CSREA Press, USA, 2003, pp. 395–401.

[139] Srivastava, B., and Koehler, J. Web service composition - current solutions and open problems. In *Proceedings of ICAPS'03 Workshop on Planning for Web Services* (Trento, Italy, 2003), 2003, pp. 28–35.

[140] Staab, S., van der Aalst, W., Benjamins, V., Sheth, A. P., Miller, J. A., Bussler, C., Maedche, A., Fensel, D., and Gannon, D. Web services: Been there, done that? *IEEE Intelligent Systems 18*, 1 (2003), 72–85.

[141] Sycara, K., Paolucci, M., Ankolekar, A., and Srinivasan, N. Automated discovery, interaction and composition of semantic web services. *Journal of Web Semantics 1*, 1 (2003), 27–46.

[142] Szyperski, C. *Component Sotfware: Beyond Object-Oriented Programming*. Addison-Wesley, New York, 1998.

[143] Szyperski, C. Component technology - what, where, and how? In *Proceedings of the 25th International Conference on Software Engineering* (Portland, USA, 2003), IEEE CS Press, Los Alamitos, 2003, pp. 684–693.

[144] Taylor, D. A. *Object-Oriented Technology: A Manager's Guide*. Addison-Wesley, New York, 1991.

[145] Tsai, T.-M., Yu, H.-K., Shih, H.-T., Liao, P.-Y., Yang, R.-D., and Chou, S.-c. T. Ontology-mediated integration of intranet web services. *IEEE Computer Magazine 36*, 10 (2003), 63–71.

[146] Tsalgatidou, A., and Pilioura, T. An overview of standards and related technology in web services. *Distributed and Parallel Databases 12*, 2-3 (2002), 135–162.

[147] Tut, M. T., and Edmond, D. The use of patterns in service composition. In *Web Services, E-Business, and the Semantic Web: CAiSE 2002 International Workshop* (Toronto, Canada, 2002), Springer, Berlín, 2002, pp. 28–40.

[148] van der Aalst, W., ter Hofstede, A. H., Kiepuszewski, B., and Barros, A. Workflow patterns. *Distributed and Parallel Databases 14*, 1 (2003), 5–51.

[149] van der Aalst, W., and van Hee, K. *Workflow Management: Models, Methods, and Systems*. The MIT Press, Cambridge, 2002.

[150] Vasko, M., and Dustdar, S. An analysis of web services workflow patterns in collaxa. In *Web Services: European Conference* (Erfurt, Germany, 2004), Springer, Berlín, 2004, pp. 1–14.

[151] Verma, K., Sivashanmugam, K., Sheth, A. P., Patil, A., Oundhakar, S., and Miller, J. A. METEOR-S WSDI: A scalable P2P infrastructure of registries for semantic publication and discovery of web services. *Journal of Information Technology and Management 6*, 1 (2005), 17–39.

[152] Vidal, J. M., Buhler, P., and Stahl, C. Multiagent systems with workflows. *IEEE Internet Computing 8*, 1 (2004), 76–82.

[153] Vinoski, S. CORBA: Integrating diverse applications within distributed heterogeneous environments. *IEEE Communications Magazine 35*, 2 (1997), 46–55.

[154] Vinoski, S. WS-Nonexistent standards. *IEEE Internet Computing 8*, 6 (2004), 94–96.

[155] Vogels, W. Web services are not distributed objects. *IEEE Internet Computing 7*, 6 (2003), 59–66.

[156] Voisard, A., and Schweppe, H. Abstraction and decomposition in interoperable GIS. *International Journal of Geographical Information Science 12*, 4 (1998), 315–333.

[157] Wang, H., Haung, J. Z., Qu, Y., and Xie, J. Web services: problems and future directions. *Journal of Web Semantics 1*, 3 (2004), 309–320.

[158] Wiederhold, G. Mediators in the architecture of future information systems. *IEEE Computer Magazine 25*, 3 (1992), 38–49.

[159] Wiederhold, G. Interoperation, mediation, and ontologies. In *Proceedings of the International Symposium on 5th Generation Computer Systems, Workshop on Heterogeneous Cooperative Knowledge-Bases* (Tokyo, Japan, 1994), ICOT, Tokyo 1994, pp. 33–48.

[160] Wiederhold, G., Wegner, P., and Ceri, S. Toward megaprogramming. *Communications of the ACM 35*, 1 (1992), 89–99.

[161] Wohed, P., van der Aalst, W., Dumas, M., and ter Hofstede, A. H. Analysis of web services composition languages: The case of BPEL4WS. In *Conceptual Modeling - ER 2003* (Chicago, USA, 2003), Springer, Berlín, 2003, pp. 200–215.

[162] Yang, J. Web service componentization. *Communications of the ACM 46*, 10 (2003), 35–40.

[163] Yang, J., and Papazoglou, M. P. Web component: A substrate for web service reuse and composition. In *Advanced Information Systems Engineering: 14th International Conference* (Toronto, Canada, 2002), Springer, Berlín, 2002, pp. 21–36.

[164] Yang, J., and Papazoglou, M. P. Service components for managing the life-cycle of service compositions. *Information Systems 29*, 2 (2004), 97–125.

[165] Zeng, L., Benatallah, B., Dumas, M., Kalagnaman, J., and Chang, H. QoS-aware middleware for web services composition. *IEEE Transactions on Software Engineering 30*, 5 (2004), 311–327.