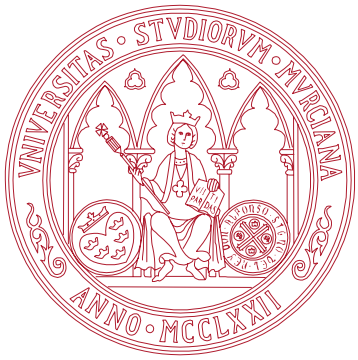

Universidad de Murcia
Facultad de Informática

Una propuesta de gestión integrada de modelos y requisitos en líneas de productos software

Tesis Doctoral

Universidad de Murcia **Facultad de Informática**

Una propuesta de gestión integrada de modelos y requisitos en líneas de productos software

Tesis Doctoral

Doctorando: Joaquín Nicolás Ros

Director: José Ambrosio Toval Álvarez

2009

Para Josefina y para las pequeñas Irene y Silvia
...y para mis padres

Agradecimientos

Quiero tener en primer lugar una mención muy especial para mis padres. He de reconocer todas las atenciones, preocupaciones y cuidados que les debo. Y en un plano más específico, les debo agradecer la oportunidad que me ofrecieron de crecer en una casa llena de libros. Sé que ambos se alegran tanto como yo de tener esta memoria en sus manos. No me quiero olvidar de mis hermanos, José Pablo y David, a los que he visto crecer hasta hacerse las personas que son ahora, de las cuales puedo estar orgulloso.

Debo tener un reconocimiento para Ambrosio Toval, con quien me inicié en la investigación hace muchos años. Justo es reconocer que es el "padre" de SIREN, la línea de trabajo base de esta tesis doctoral, que subyace a lo largo de la misma. Sus comentarios sobre el enfoque de la tesis doctoral y los sucesivos borradores de la memoria han ayudado a mejorar sustancialmente la calidad de la misma.

Quiero recordar a mis compañeros de *Pasillo 4*, todos ellos fantásticos, cada uno a su manera, Quino, Marcos, Ginés, y de forma especial a Begoña Moros, con quien he compartido muchas horas de trabajo codo con codo en el desarrollo de SIREN, y a María José Ortín, con quien compartí muchos años de despacho. Espero que ambas disfruten de la etapa que se abre en sus vidas con sus nuevos retoños.

En este momento recuerdo también otras personas con las que he compartido lugar en el grupo de investigación o en el departamento y que me han mostrado que "generosidad" no es solamente una palabra en el diccionario. Con algunos ya no tengo contacto y nunca leerán estas líneas, pero siguen teniendo todo mi aprecio. En particular quiero mencionar a Joaquín Lasheras, con quien he compartido horas de trabajo y algunas vivencias, y a Francisco J. Lucas y Fernando Molina. A todos ellos les deseo toda la suerte del mundo en su nueva etapa profesional. Quiero también expresar mi agradecimiento a los alumnos de los proyectos fin de carrera que han ayudado a desarrollar aspectos concretos de la tesis.

Mi agradecimiento también a mis amigos, con la mayoría de los cuales suelo hablar poco de estas cosas, pero que también han ayudado, aunque no lo sepan.

Y finalmente, quiero terminar agradeciendo de forma muy especial a Josefina su apoyo y su comprensión ante la dedicación que he prestado a la Universidad durante estos años tan intensos para nosotros en tantos ámbitos. Sin su ayuda esta tesis doctoral no habría sido posible. Y por último, qué puedo decir de mis pequeñas, Irene y Silvia... tal vez simplemente que son la alegría de mi vida. Agotadoras, su curiosidad innata ante un mundo completamente nuevo es comparable a la de los mejores investigadores.

Sólo hay mundo donde hay lenguaje

Martin Heidegger

Índice de contenidos

Índice de figuras

Índice de tablas

Resumen

En esta tesis doctoral se presenta una propuesta de Ingeniería de Requisitos (IR) para líneas de productos que integra modelos de análisis del dominio y requisitos en lenguaje natural. Esta propuesta se construye incrementalmente en tres fases: (1) se estudia la reutilización de requisitos textuales, definiendo un método de IR basado en reutilización de requisitos en lenguaje natural, denominado SIREN (*SImple REuse of RequiremeNts*), validado en entornos industriales, que incluye un modelo de referencia de requisitos, un conjunto de técnicas y de guías, un modelo de procesos y una herramienta de soporte. Se propone también una extensión de SIREN para el desarrollo global de software, denominada SIRENgsd (donde *gsd* procede de *global software development*). SIRENgsd se presenta junto con un repositorio de amenazas y salvaguardas para la IR cuando esta se lleva a cabo en entornos globalizados, repositorio que procede de una revisión sistemática de la literatura; (2) se describe SIRENspl (donde *spl* procede de *software product line*), una evolución de SIREN con el objetivo de modelar el dominio de una línea de productos, los sistemas teleoperados para mantenimiento de cascos de buques (STO). SIRENspl incorpora técnicas de análisis del dominio seleccionadas y particularizadas a partir de un estudio del estado del arte en IR para líneas de productos, e incluye soporte automatizado específico; y finalmente (3) se plantea el interés de la integración de modelos de ingeniería del software con especificaciones de requisitos en lenguaje natural, en este orden, es decir, implicando la generación de requisitos textuales a partir de modelos de ingeniería del software. El interés de este enfoque se corrobora a través de una revisión sistemática de la literatura, tras la cual se define lo que se denomina una correspondencia de *aplanamiento* de modelos de análisis del dominio (procedentes del enfoque de SIRENspl) a requisitos en lenguaje natural (procedentes del enfoque de SIREN). Para evaluar la viabilidad de esta propuesta de aplanamiento se realiza una implementación de la misma utilizando técnicas de transformación de modelos, mediante una descripción formal de los modelos de inicio y destino y el uso de un lenguaje de transformaciones declarativo. Finalmente la propuesta de aplanamiento se valida con su aplicación retrospectiva a los modelos resultado del caso de estudio de los STO.

Palabras clave

Ingeniería de Requisitos; Requisitos en Lenguaje Natural; Reutilización de Requisitos; Ingeniería de Requisitos para Desarrollo Global de Software; Ingeniería de Requisitos para Líneas de Productos; Generación de Requisitos desde Modelos de Ingeniería del Software.

Abstract

In this doctoral thesis an approach to requirements engineering (RE) for product lines integrating domain analysis models and natural language requirements is formulated. This proposal is incrementally constructed on three levels: (1) natural language requirements reuse is studied, and then the SIREN (*SImple REuse of RequiremeNts*) method is defined. SIREN is an RE method based on natural language requirements reuse, which has been validated in industrial environments. SIREN includes a requirements reference model, a set of techniques and guidelines, a process model, and tool support. Furthermore, an extension of SIREN called SIRENgsd (where *gsd* stands for *global software development*) is proposed, which extends SIREN to the global software development. SIRENgsd is presented together with a risks and safeguards repository concerning RE when performed in distributed environments. This repository is drawn from a systematic review of the literature on this topic; (2) an evolution of SIREN is proposed, SIRENspl (where *spl* stands for *software product line*), with the intent of modeling the teleoperated systems for ship hull maintenance (TOS). SIRENspl encompasses tailored, state-of-the-art domain analysis techniques and includes specific tool support; and (3) the interest of the generation of textual requirements starting from software engineering models is corroborated through a systematic review of the literature. A *flattening* correspondence is defined which maps the SIRENspl domain analysis models to SIREN natural language requirements. To assess the viability of this proposal an implementation is made which uses model transformation techniques, by means of formal descriptions of the initial and target models and a declarative transformation language. Finally the proposal is validated through its retrospective application to the models developed in the TOS case study.

Keywords

Requirements Engineering; Natural Language Requirements; Requirements Reuse; Requirements Engineering for Global Software Development; Requirements Engineering for Product Lines; Textual Requirements Generation from Software Engineering Models.

1 Introducción, análisis de objetivos y metodología

1.1 Introducción

Hoy día se reconoce ampliamente la necesidad de un enfoque disciplinado en el desarrollo de software y existe un cierto consenso en cuanto a que la Ingeniería del Software ha ganado madurez. Sin embargo, como afirma Sommerville [275], la explosión tecnológica que se viene produciendo desde la segunda mitad de los años noventa, y que por ejemplo se puede observar en el auge de múltiples tecnologías de componentes distribuidos y en la evolución de las comunicaciones, no ha sido seguida por una evolución de la misma magnitud en la práctica de la ingeniería del software. Por ejemplo, la industria ha venido utilizando con demasiada frecuencia el modelo de procesos en cascada [173], a pesar de que sus serios problemas son conocidos y documentados desde hace muchos años. Muchas organizaciones siguen sin aplicar adecuadamente las técnicas de ingeniería del software, con lo que demasiados proyectos producen software no confiable, entregado tarde y con desviaciones presupuestarias [276]. Todavía se observa en la práctica y se refleja en la literatura de ingeniería del software la necesidad de incrementar la calidad del producto software y la productividad en el proceso de desarrollo del mismo. Son éstas dos cuestiones muy complejas que se deben abordar desde diversos puntos de vista, ya que como afirma Brooks en un célebre y clásico artículo [46], “no existe bala de plata”.

Con esta tesis doctoral se trata de contribuir en un aspecto concreto del desarrollo de software, a través de una propuesta de gestión de requisitos en líneas de productos software que integra modelos y requisitos textuales. Con este trabajo se trata de acercar el estado de la investigación y el estado de la práctica en Ingeniería de Requisitos (IR), y en particular en IR para líneas de productos. Por un lado, el estado del arte en IR se ha centrado en modelos de requisitos, y en el marco particular de la IR para líneas de productos, en modelos que representan de un modo u otro la variabilidad de la línea de productos. Por otro lado, en la práctica todavía muchas personas involucradas en algún sentido en el desarrollo de software prefieren la documentación en lenguaje natural de los requisitos. Así pues el estado de la investigación y el estado de la práctica han estado tradicionalmente muy separados en ingeniería del software en general [259] y en IR en particular [158].

El interés de este trabajo se justifica en la importancia de la IR en el desarrollo de software, y en la conveniencia de gestionar de forma integrada modelos de requisitos y especificaciones de requisitos textuales, generando documentación de requisitos en lenguaje natural de forma sincronizada con los modelos de requisitos desarrollados. Esta propuesta se fundamenta también en la necesidad de mejorar la reutilización del software, y en particular la reutilización de requisitos, como vía necesaria para mejorar la calidad y productividad en el desarrollo de software. En este sentido, el paradigma de líneas de productos ha mostrado en los últimos años la posibilidad de mejorar la reutilización en dominios verticales de aplicación. Con el objetivo de proporcionar un contexto para el establecimiento de la hipótesis de partida y de los objetivos de esta tesis doctoral, en el resto de este apartado se justifican brevemente todos estos postulados.

1.1.1 El proceso de ingeniería de requisitos en el desarrollo de software

Según la definición de Kotonya y Sommerville [165], el término *ingeniería de requisitos* (IR) implica el uso de procedimientos repetibles y sistemáticos para asegurar la obtención de un conjunto de requisitos relevante, completo, consistente y fácilmente comprensible y analizable por parte de los diferentes actores implicados en el desarrollo del sistema. Un requisito es, según la definición del IEEE 610 [137]: (a) una condición o capacidad necesitada por un usuario para resolver un problema o alcanzar un objetivo; (b) una condición o capacidad que un sistema o un componente de un sistema debe satisfacer o poseer de acuerdo con un contrato, estándar, especificación u otro documento impuesto formalmente; o bien (c) una representación documentada de una condición o capacidad como en (a) o (b).

El objetivo básico de la IR es la especificación de qué debe hacer un sistema y de las restricciones de diseño que condicionan cómo ha de ser implementado, con el objetivo de desarrollar un software correcto, es decir, un software que funcione como los clientes desean que funcione. Así pues parece evidente que la construcción de un nuevo sistema o la modificación de uno existente se debería realizar sobre la base de un conocimiento preciso de lo que se debe desarrollar o cambiar. Sin embargo, según nuestra experiencia, con frecuencia el proceso de IR se ha realizado –cuando se ha realizado– de forma inapropiada, de manera que la especificación de requisitos se ha reducido, en muchas ocasiones, a una simple declaración de objetivos genéricos en unas pocas páginas. En este sentido, Neill y Laplante [221] afirman en un estudio sobre la práctica de la IR que "el 52% de los encuestados no considera que su organización haga suficiente IR". Todo ello a pesar de que desde hace años numerosos estudios empíricos respaldan la afirmación de que la IR es un proceso crítico en el desarrollo de software correcto (sólo por mencionar unos pocos, véase [109, 156, 203]). Por ejemplo, en un análisis que implica más de 600 compañías de software de Estados Unidos, Europa y Japón, Blackburn *et al.* [43] concluyen que "(…) con respecto al tiempo utilizado en las diversas etapas del desarrollo de software, la única etapa con una correlación positiva es la del tiempo empleado en la determinación de los requisitos de los clientes: las empresas más productivas dedicaban significativamente más tiempo a estas tareas en sus procesos de desarrollo". Estos autores también afirman que el tiempo y el esfuerzo dedicado al prototipado y a otras técnicas para refinar los requisitos de los clientes son amortizados rápidamente en un ciclo de desarrollo más corto.

Glass [116] afirma que "una de las causas más comunes de proyectos descontrolados es la inestabilidad de los requisitos". Glass destaca este hecho como la causa más frecuente de proyectos fallidos, junto a "realizar una inadecuada estimación del proyecto en tiempos y costes". El origen de la inestabilidad en los requisitos radica, en la mayoría de las ocasiones, en que el cliente o los usuarios no conocen realmente cuáles son sus necesidades. Para Glass, como para numerosos autores, como Pressman [250], "los errores relacionados con los requisitos son los más caros de corregir durante la construcción del software". En este sentido, según Glass, "el problema más difícil de corregir, relacionado con los requisitos, es que no sean descubiertos a tiempo requisitos que son relevantes para el proyecto". Un hecho adicional es que "al pasar de los requisitos del problema a los requisitos del diseño, se produce una explosión de *requisitos derivados* (requisitos que surgen para satisfacer una determinada solución de diseño) causada por la complejidad del proceso de la solución", lo cual según Glass

conlleva que a veces la lista de estos nuevos requisitos derivados sea hasta 50 veces más grande que la de los requisitos originales.

En nuestra opinión, algunos de los problemas enunciados anteriormente se deben a que la IR adolece de un problema intrínseco de comunicación entre todas aquellas personas que tienen un interés determinado en un desarrollo de software (los *stakeholders*, a los que en esta memoria nos referiremos simplemente como los *interesados*). Por un lado, por ejemplo, tenemos a los directivos de distintos niveles de la eventual organización cliente, a los expertos en mercadotecnia y a los usuarios finales del software, y por otro lado tenemos a los responsables del proyecto y a los desarrolladores. El ingeniero de requisitos debe ser capaz de jugar un difícil papel intermedio y especificar todos los requisitos de manera que se transmita al desarrollador de forma completa y no ambigua lo que el cliente desea. Esto implica no sólo especificar los requisitos que los clientes son capaces de transmitir por sí mismos, sino también ayudarles a desarrollar sus requisitos, a extraer necesidades que inicialmente no sospechaban. Una mala gestión en la identificación de los interesados incide de forma especialmente negativa en que no sean descubiertos a tiempo requisitos que son relevantes para el proyecto. Además, en ocasiones, es preciso negociar y adoptar decisiones de compromiso (*trade-offs*) para reconciliar distintas necesidades y expectativas de los interesados.

Durante el proceso de IR se realizan de forma cíclica las actividades de (1) identificación y consenso (*elicitation*, *extracción*); (2) análisis y negociación; (3) documentación (especificación); y (4) validación de requisitos. Con frecuencia se distingue también una actividad de gestión de requisitos propiamente dicha [275] que se refiere a la programación, coordinación y documentación de las actividades anteriores, controlando los cambios sobre los requisitos. En adelante, cuando se hable de gestión de requisitos se hará con este significado concreto y no con el significado más general que prefieren algunos autores y que equipara este término a IR.

La IR influye decisivamente en el resto de procesos de desarrollo y mantenimiento de un sistema. No es un proceso que se realiza sólo en las primeras etapas del desarrollo de un proyecto y produce un conjunto de documentos que quedan “congelados” [158, 174]. Dado que los requisitos son de naturaleza cambiante, de forma inevitable se encuentran sujetos a cambios constantes durante todo el desarrollo: es por tanto necesario gestionar adecuadamente los cambios en los requisitos a lo largo del desarrollo, la explotación y el mantenimiento del software, a través sus relaciones de traza con el resto de los productos de desarrollo y con los compromisos que se hayan establecido. De forma complementaria, los requisitos constituyen una buena base sobre la que establecer métricas sobre el proyecto y sobre el producto del desarrollo, y para derivar la especificación de los casos de prueba que se han de realizar sobre el sistema o producto final.

Son muchas las referencias que proporcionan una buena visión de la IR. Entre ellas podemos destacar los textos de Wiegers [296], Robertson y Robertson [265] y Davis [82], que presentan tres visiones prácticas y vigentes en relación con esta disciplina. En la Sección 2.2 de esta memoria se proporciona una breve descripción general de la IR.

1.1.2 Reutilización e ingeniería de requisitos

Actualmente siguen siendo válidas las ideas que Brooks expresara sobre la Ingeniería del Software hace veinte años [46], en tanto que dentro de esta disciplina existen aproximaciones que se dirigen a la complejidad *accidental* del software, mientras que otras se dirigen a la complejidad *esencial* del mismo. Entre estas últimas, y en conexión con el papel que desempeña la IR en el desarrollo de software, comentado en el apartado anterior, Brooks [46] considera el refinamiento de requisitos y el prototipado rápido: "La parte más ardua de la construcción de un sistema software es decidir de forma precisa qué construir. Ninguna otra parte del trabajo conceptual es tan difícil como establecer los requisitos técnicos detallados, incluyendo las interfaces con las personas, las máquinas y otros sistemas software. Ninguna otra parte del trabajo condiciona tanto el sistema resultante si se hace mal. Ninguna otra parte es más difícil de rectificar después."

Entre las estrategias de ingeniería del software que atacan la esencia conceptual del problema, Brooks cita también *comprar en lugar de construir* (*buy versus build*), una idea que creemos que se puede extrapolar a una forma de reutilización del software. La reutilización del software es para Meyer [214] "la capacidad de los elementos software de servir para la construcción de muchas aplicaciones diferentes". Este autor considera la reutilización uno de los factores externos de calidad del software, y resume los beneficios esperados de la reutilización en (1) una mejora de la oportunidad de los desarrollos (mejoras en los tiempos para llevar los proyectos hasta su culminación y los productos hasta el mercado); (2) una disminución de los esfuerzos de mantenimiento; (3) una mejora en la fiabilidad, eficiencia y consistencia del software desarrollado; y (4) como inversión (hacer reutilizable el software ayuda a preservar el *saber hacer –know how–* y las creaciones de los mejores desarrolladores, transformando un recurso frágil en un valor permanente dentro de la organización de desarrollo).

Mili *et al.* [216] afirman que "la reutilización del software es la única aproximación realista para conseguir las ganancias de calidad y productividad que la industria del software necesita". A la luz de las ideas de Brooks, diríamos que se trata de una aproximación necesaria aunque no suficiente, considerando que no existe la deseada "bala de plata". En esta misma línea, cuando examina "el camino por recorrer" en la ingeniería del software, Pressman [250] concluye que "la reutilización y la ingeniería del software basada en componentes ofrecen la mejor oportunidad en cuanto a mejoras en la magnitud de la calidad de los sistemas y en el tiempo en que llegan al mercado".

Existe una corriente dentro de la reutilización del software que considera reutilizable cualquier artefacto software producido durante el desarrollo, ya sea la especificación del sistema o del producto, diseños, código fuente, casos de prueba, planes del proyecto, planes de calidad, etc. De este modo con el término *asset* se nombra cualquier elemento software reutilizable. En este contexto, desde mediados de los noventa la reutilización de especificaciones o de requisitos se ha venido postulando por parte de numerosos autores como una vía prometedora (y menos explorada que la reutilización de código o de diseños) para ayudar a conseguir las ganancias de calidad y productividad que el desarrollo de software necesita. Existe un cierto consenso (véase, por ejemplo, los trabajos de Sommerville [275] o de Cybuslky y Reed [67]) en que los beneficios de la reutilización son mayores cuando se incrementa el nivel de abstracción, y no se reutiliza sólo código sino también diseño y especificaciones. En este sentido, Favaro [99] afirma

que “un requisito bien formulado, cuantificado, y reutilizable (…) es tan valioso como un módulo de software reutilizable”. Además Robertson y Robertson [265] postulan que comenzar con un conjunto de requisitos que han sido especificados para otros proyectos o dominios sirve para mejorar la precisión de la especificación de requisitos y para reducir el tiempo para elaborar esta especificación. Sin embargo, en la medida de nuestro conocimiento, son Rine y Nada [262] los únicos que han mostrado empíricamente que el nivel de reutilización determina la efectividad de las mejoras en productividad, calidad y tiempo de desarrollo, concluyendo que se obtienen beneficios mayores al considerar la reutilización durante los procesos iniciales del ciclo de vida de desarrollo del software. En este sentido, cuando a finales del siglo pasado y principios de este examinaban el camino por recorrer en la investigación en IR, reconocidos autores como Nuseibeh y Easterbrook [231] y Zave [302] establecían que la reutilización de modelos de requisitos constituía uno de los principales desafíos de la IR. Estos autores afirmaban que se esperaba que se desarrollaran modelos de referencia de requisitos en muchos dominios de aplicación, de manera que se redujera drásticamente el esfuerzo de desarrollo de nuevas especificaciones de requisitos, facilitando además la selección de paquetes de software (COTS, *Comercial Off-The-Shelf Software*). En este contexto se pueden situar los trabajos clásicos sobre análisis del dominio de Prieto-Díaz y Arango [251] o los posteriores patrones de análisis de Fowler [108]. Para Clements y Northrop [57], el análisis del dominio es “un proceso para capturar y representar información acerca de las aplicaciones en un dominio, específicamente características comunes y razones para la variabilidad”. Para estos autores, un dominio es simplemente “un área de conocimiento o actividad caracterizada por un conjunto de conceptos y terminología comprendida por los profesionales en dicha área”.

Una voz en cierto sentido discordante con esta corriente de reutilización de requisitos es la de Meyer [214], que distingue los siguientes niveles de reutilización del software: reutilización de personal, de diseño y especificaciones, patrones de diseño, código fuente y módulos abstractos. Para Meyer es dudosa la noción misma de un diseño o especificación como un artefacto software con vida propia, separada de la implementación correspondiente: si así fuera sería difícil garantizar que la especificación, el diseño y la implementación se mantuvieran compatibles a lo largo de la evolución de un sistema software, de modo que si sólo se reutiliza la especificación o el diseño se corre el riesgo de reutilizar elementos software incorrectos u obsoletos. Meyer valora negativamente la “estrecha visión” de que sólo tienen interés “los aspectos verdaderamente difíciles” de la reutilización, es decir “la reutilización de diseños y especificaciones”, mientras que la reutilización de código es “trivial”. Comenta Meyer que la orientación a objetos y la aparición de componentes reutilizables bastaron para “derrotar” esta visión, si bien la idea de reutilización de diseños cobra más interés si la distancia entre módulos y diseños desaparece y se difumina así la distinción entre reutilizar diseños y reutilizar módulos.

La reutilización de requisitos es por tanto un enfoque que ha merecido el interés de la literatura, en la cual muchos estudios postulan sus beneficios (porque los formulan a priori, pero no los demuestran empíricamente), aunque también existen otros trabajos en los que se desconfía de esta corriente. En la Sección 2.3 de esta memoria se proporciona un estudio más detallado del estado del arte en reutilización de requisitos.

1.1.3 Líneas de productos software: ingeniería de requisitos del dominio y de la aplicación

En el campo de la reutilización del software, las líneas de productos software surgieron a mediados de los años noventa como una de las aproximaciones más prometedoras. En los últimos años este paradigma de desarrollo ha ganado madurez y se ha asentado en determinados dominios de aplicación, como automoción y comunicaciones. Las líneas de productos software permiten incrementar la reutilización en determinados dominios de aplicación verticales o "nichos" del mercado. En la definición clásica de Clements y Northrop [57] una línea o familia de productos (en esta memoria ambos términos se consideran sinónimos) viene dada por un conjunto de sistemas que comparten características comunes que satisfacen las necesidades específicas de un segmento del mercado, desarrollados a partir de un conjunto preestablecido de *assets* o elementos software reutilizables. Para Lam *et al.* [171], una de las diez cuestiones clave para mejorar la reutilización de requisitos es precisamente adoptar una aproximación de línea de productos.

La discusión de la sección anterior acerca del nivel de la reutilización tiene una respuesta clara en el campo de las líneas de productos. Para van der Linden *et al.* [288], a diferencia de muchas estrategias tradicionales de reutilización, que se centran en la reutilización de assets de código, el enfoque basado en líneas de productos engloba todos los assets que son relevantes a través del ciclo de vida de desarrollo de software.

El desarrollo de líneas de productos se divide usualmente en dos procesos complementarios: la ingeniería del dominio y la ingeniería de la aplicación (o del producto). Los requisitos de una línea de productos definen los productos de dicha línea y sus características comunes y variables. La IR para líneas de productos debe gestionar los requisitos de la línea de productos y los requisitos de los productos concretos de la línea. La IR para líneas de productos debe incorporar un mecanismo mediante el cual el conjunto de requisitos para un producto concreto sea producido de manera fácil y rápida a partir de los requisitos de la línea de productos. Sin embargo, como afirman Käkölä y Dueñas [159], la investigación en IR para líneas de productos se ha centrado tradicionalmente en la IR del dominio (usualmente denominada "análisis del dominio" o "modelado del dominio"), y ha descuidado la IR del producto.

Siguiendo el enfoque inicial de Greenfield y Short sobre fábricas (factorías) de software (*software factories*) [125], en las modernas fábricas de software [246] la IR se orienta hacia líneas de productos. El reto fundamental de la IR para líneas de productos es el tratamiento adecuado de la *variabilidad de los requisitos* [249]. La IR para líneas de productos se relaciona habitualmente con el uso de modelos de características [160], con extensiones para los modelos de casos de uso (por ejemplo, ver [155]), y más recientemente con *modelos de variabilidad independientes* [37, 47]. En contraste, la IR *convencional* se suele orientar a la gestión de requisitos textuales –preferentemente en lenguaje natural– y de documentos de requisitos. Los conceptos y técnicas generales de la gestión de requisitos convencional, sin embargo, son también de utilidad en el campo de las líneas de productos.

Para conocer una visión actual del estado de la práctica en IR para líneas de productos se puede acudir a Pohl *et al.* [249], que recogen la experiencia de tres proyectos ESPRIT consecutivos: ESAPS (1999-2001), CAFÉ (2001-2003) y FAMILIES (2003-

2005). Es también destacable la propuesta de Gomaa [122], que muestra un método práctico de gestión de requisitos para líneas de productos basado en UML. van der Linden *et al.* [288] proporcionan otra visión completa de la práctica en este campo. Para conocer el estado actual de la investigación en líneas de productos un texto reciente e interesante es el de Käkölä y Dueñas [159]. En la Sección 3.3 de esta memoria se proporciona una breve descripción de la IR para líneas de productos.

1.1.4 Generación de especificaciones de requisitos desde modelos de ingeniería del software

En un estudio sobre futuras direcciones de investigación en IR, Cheng y Atlee [70] afirman que se ha trabajado poco en la interconexión de varios tipos de modelos de requisitos, y que se requiere investigar cómo integrar técnicas de IR de manera que puedan ser utilizadas de forma sinérgica. De acuerdo con Goldsmith [121], a pesar de que en la comunidad de ingeniería del software está ampliamente asumido que los modelos y/o los prototipos ejecutables constituyen el medio más apropiado para capturar y comunicar los requisitos, los requisitos deben ser escritos en palabras para que puedan ser revisados adecuadamente. Cuando Davis [82] reflexiona sobre la mejora del proceso de gestión de requisitos, concluye que el uso combinado de modelos de ingeniería del software y de listas de requisitos textuales constituye una buena práctica para mejorar dicho proceso de gestión de requisitos, pues permite obtener los beneficios de ambas aproximaciones. Por un lado, normalmente las técnicas de modelado son expresivas, precisas, y facilitan la especificación y la comprensión de los requisitos por parte del equipo de desarrollo. Por otro lado, las listas de requisitos en lenguaje natural proporcionan un contrato entre clientes y desarrolladores y simplifican la gestión de los requisitos del proyecto, clarificando el tamaño del mismo y el estado actual de desarrollo de los requisitos, y facilitando la validación por parte de los clientes.

La práctica habitual en ingeniería del software ha sido la especificación de modelos del sistema o del software a partir de requisitos escritos en lenguaje natural. Sin embargo, si se proporcionara la posibilidad de extraer información de los modelos de desarrollo del sistema o del software (que usualmente tienen una representación gráfica) en forma de requisitos textuales (que usualmente se escriben en lenguaje natural), de forma automática o fuertemente asistida, o al menos se proporcionaran guías claras para integrar modelos y requisitos textuales, se facilitaría el uso combinado de modelos de ingeniería del software y de requisitos textuales, y se obtendrían ciertos beneficios adicionales:

- Reducción en el esfuerzo de escribir los requisitos. Es ampliamente aceptado que un requisito debe ser no ambiguo, completo, consistente y verificable (ver por ejemplo el estándar IEEE 830 [138]). Escribir una especificación de requisitos que satisfaga tales criterios es una tarea meticulosa que puede requerir un esfuerzo considerable. La derivación automática o fuertemente asistida de requisitos textuales a partir de modelos puede ayudar a conseguir ganancias de productividad en el proceso de especificación de requisitos.
- Mejora en la completitud de la especificación de requisitos. La generación de parte de los requisitos de manera automática o fuertemente asistida contribuye a la completitud de la especificación de requisitos, pues para los interesados es más sencillo aceptar o refinar requisitos que formularlos desde un principio. En

opinión de Maiden *et al.* [196], para una persona es más fácil identificar errores de comisión que errores de omisión.

- Posibilidad de automatizar las trazas entre modelos y requisitos textuales. CMMI (*Capability Maturity Model Integration*) [58, 59] recomienda mantener trazabilidad bidireccional entre los requisitos y los productos del desarrollo. Dentro del área de proceso de Gestión de Requisitos (*Requirements Management*) del Nivel 2 de CMMI (ver Sección 2.2.2), se recomienda específicamente el mantenimiento de "trazabilidad bidireccional (entre requisitos y productos del trabajo)" (SP 1.4).

Se trata por tanto de estudiar la obtención de requisitos textuales a partir de modelos de ingeniería del software, en este sentido, de modelos a requisitos. En la Sección 4.2 de esta memoria se muestra una revisión sistemática de la literatura sobre este tema. El problema inverso, es decir la generación de modelos a partir de los requisitos textuales, no se investiga en esta tesis doctoral. Es este un problema distinto, al que los desarrolladores se han enfrentado tradicionalmente de forma *ad hoc*, y en el que también se requiere más investigación. Tradicionalmente en la práctica de la ingeniería del software se obtienen de forma no sistemática modelos del sistema a partir de especificaciones de requisitos en lenguaje natural.

1.2 Hipótesis y objetivos

La hipótesis de partida de esta tesis doctoral es que es factible definir una propuesta de IR para líneas de productos que facilite la reutilización de requisitos y que integre modelos de ingeniería del software y requisitos textuales. El fin último de esta propuesta es la mejora de la calidad de las especificaciones de requisitos de líneas de productos y de la productividad en el proceso de especificación, si bien no se prevé que tales mejoras en la calidad y la productividad puedan ser medidas experimentalmente de forma cuantitativa en el ciclo de desarrollo de software, sino sólo de forma cualitativa, debido a la ausencia de registros previos en los casos de estudio involucrados en esta investigación.

Para probar la hipótesis anterior, se define el siguiente objetivo global para esta tesis doctoral:

> *Objetivo Global*. Definir una propuesta de gestión de requisitos para líneas de productos que integre modelos de ingeniería del software y requisitos textuales en lenguaje natural.

Este objetivo global se desglosa en los siguientes objetivos parciales:

> *Objetivo 1*. Estudiar la reutilización de requisitos textuales, definiendo un método de IR basado en reutilización que incluya un metamodelo de requisitos, un conjunto de técnicas, un modelo de procesos y una herramienta de soporte.

Objetivo 2. Definir un metamodelo de variabilidad en una línea de productos software, que sea compatible con los resultados del Objetivo 1, y establecer un conjunto de técnicas que representen dicho modelo de variabilidad en el nivel de los requisitos.

Objetivo 3. Definir un proceso de generación de documentación de requisitos textuales, en un proceso de desarrollo de software iterativo e incremental, que sincronice los modelos definidos en el Objetivo 2 con el metamodelo de requisitos textuales establecido en el Objetivo 1.

Objetivo 4. Diseñar e implementar el prototipo de una herramienta que soporte los modelos definidos en 2 y el proceso de generación definido en 3.

Objetivo 5. Validar los resultados anteriores en al menos un caso de estudio real.

1.3 Metodología de la investigación

En un análisis de la literatura de ingeniería del software, Glass *et al.* [116] muestran que tradicionalmente el método de investigación mayoritario en ingeniería del software ha sido el análisis o estudio conceptual, concluyendo que "los investigadores en ingeniería del software tienden a analizar e implementar nuevos conceptos, y hacen muy poco más". Por el contrario, en la realización de esta tesis doctoral se han utilizado varios métodos de investigación: casos de estudio, investigación en acción y revisiones sistemáticas de la literatura. En este apartado se introduce brevemente cada uno de estos métodos.

1.3.1 Casos de estudio

Esta sección se basa en un destacado trabajo de Runeson y Höst [266], que presentan un conjunto de cuestiones que contribuyen a la calidad de una investigación en ingeniería del software basada en *estudios de casos* (*case studies*), a los que generalmente la comunidad de ingeniería del software se refiere con el término de "casos de estudio". Estos autores compilan guías para el investigador que conduce un caso de estudio, para el revisor de borradores basados en casos de estudio y para el lector de contribuciones basadas en casos de estudio. Partiendo de este trabajo, podemos caracterizar de la siguiente manera la investigación basada en casos de estudio: (1) se trata de un método de investigación empírico que investiga un fenómeno en su contexto real donde, en ocasiones, el límite entre el fenómeno a estudiar y su contexto no está perfectamente delimitado; (2) se utilizan distintas fuentes de evidencia; y se recoge información de personas, grupos y organizaciones; y (3) no existe un completo control del experimento.

Los casos de estudio han sido utilizados durante mucho tiempo en ciencias sociales como Psicología, Sociología, Ciencias Políticas, Trabajo Social y Ciencias Empresariales. En estas disciplinas los casos de estudio se conducen con el objetivo de

incrementar el conocimiento sobre fenómenos políticos y sociales relacionados con individuos, grupos y organizaciones. Más recientemente, los casos de estudio se vienen utilizando en Sistemas de Información, un área de conocimiento en la cual los casos de estudio son considerados un método de investigación mucho más maduro que en Ingeniería del Software. En Ingeniería del Software, el método basado en casos de estudio ha sido considerado tradicionalmente de menor valor que otros métodos basados en experimentos controlados, pues se suele considerar que el desarrollo de los casos de estudio está sesgado por los investigadores que los llevan a cabo y que no es posible generalizar conocimiento a partir de ellos.

El desarrollo de software es llevado a cabo por individuos, grupos y organizaciones y las cuestiones políticas y sociales son de importancia en este desarrollo. Por tanto, la Ingeniería del Software parece una disciplina en la que la investigación basada en casos de estudio puede ser adecuada si se realiza con rigor. Las características de los objetos de estudio de Ingeniería del Software son diferentes de los objetos de estudio de las ciencias sociales y en cierta medida también de los objetos de estudio de Sistemas de Información. En Ingeniería del Software los objetos de estudio son (1) corporaciones privadas u organismos públicos que *desarrollan* software en lugar de corporaciones privadas u organismos públicos que *usan* sistemas software; (2) estos objetos de estudio están orientados a proyectos más que a funciones; y (3) el trabajo estudiado es un trabajo de ingeniería avanzado en lugar de un trabajo rutinario. Estas tres cuestiones dificultan la aplicación en Ingeniería del Software del método de investigación basado en casos de estudio. Sin embargo, la investigación basada en casos de estudio es adecuada en muchas áreas de la Ingeniería del Software, en la cual los objetos de estudio son fenómenos contemporáneos difíciles de estudiar aisladamente. Los casos de estudio son particularmente apropiados cuando en un entorno contemporáneo se busca comprender cómo y porqué la tecnología es usada o no usada, funciona o no funciona, y cuando se tiene poco o ningún control sobre las variables. El estudio de fenómenos en su contexto, particularmente cuando el límite entre el fenómeno a estudiar y su contexto es poco claro, es particularmente útil en Ingeniería del Software. La experimentación en Ingeniería del Software ha mostrado, por ejemplo al tratar de replicar estudios, que existen muchos factores que inciden en el desarrollo de una actividad de ingeniería del software. La investigación en Ingeniería del Software no se suele desarrollar en un laboratorio en el que todos los factores pueden ser controlados.

Tres grandes métodos de investigación, relacionados con el basado en casos de estudio, pero de distinta naturaleza, son los siguientes:

- *Survey* (*sondeo*, *inspección*), que consiste en “la colección de información estandarizada a partir de una población específica, o de una muestra de una población específica, normalmente, aunque no necesariamente, por medio de cuestionarios o entrevistas”.
- *Experimento*, o *experimento controlado*, que se caracteriza por “medir en una variable los efectos de manipular otra variable” y en que los sujetos son asignados a los tratamientos de forma aleatoria. En los *casi-experimentos* (*quasi-experiments*), los sujetos no se asignan de forma aleatoria a los tratamientos.
- *Investigación en acción* (I-A), que se trata con más profundidad en la Sección 1.3.2, cuyo propósito es influir o cambiar algún aspecto en el objeto de la investigación. La investigación basada en casos de estudio es puramente

observacional, mientras que I-A está enfocada e implicada en un proceso de cambio en el objeto de estudio.

En el marco de una terminología que no está formalmente establecida, para Runeson y Höst los *estudios de campo* (*field studies*) son combinaciones de varios casos de estudio, mientras que los *estudios etnográficos* se relacionan con prácticas culturales (estos últimos consisten generalmente de estudios de larga duración con grandes cantidades de datos). Por otro lado, Perry *et al.* [242, 243] distinguen entre dos conceptos que con frecuencia se confunden: *informe de experiencia* (*experience report*) y caso de estudio. El primero es un informe retrospectivo sobre una experiencia (típicamente industrial) que ha sido particularmente interesante, y de la cual se extraen lecciones aprendidas. Por contra, los casos de estudio siempre comienzan con una cuestión a investigar y en ellos siempre se recogen y analizan datos para responder la cuestión inicial.

Un caso de estudio puede contener elementos de otros métodos de investigación. Por ejemplo: (1) dentro de un caso de estudio se puede conducir un survey; (2) habitualmente, un caso de estudio es precedido por una revisión de la literatura; (3) para recoger datos en casos de estudio se utilizan métodos etnográficos, como entrevistas y observaciones.

Los datos recogidos en un estudio empírico pueden ser *cuantitativos* o *cualitativos*. Los datos cuantitativos engloban números y clases, mientras que los datos cualitativos engloban palabras, descripciones, figuras, diagramas, etc. Los datos cuantitativos se analizan mediante estadísticas, mientras que los datos cualitativos se analizan estableciendo categorías. Los casos de estudio se suelen basar en datos cualitativos: un caso de estudio no proporciona conclusiones con significado estadístico, pues la selección de sujetos y de objetos de estudio no se basa en muestras estadísticamente representativas. En cambio, los resultados de la investigación se obtienen a través del análisis en profundidad de casos típicos o especiales. Con frecuencia una combinación de datos cualitativos y cuantitativos proporciona la mejor comprensión del fenómeno estudiado (a veces, a estos métodos se les denomina “métodos mixtos”).

Un proceso de investigación puede ser caracterizado como *fijo* o *flexible*. En un proceso fijo, todos los parámetros se definen al comienzo del estudio, mientras que en un proceso de diseño flexible ciertos parámetros clave pueden ser cambiados durante el curso del estudio. En la Tabla 1 se contrastan ciertas características esenciales de los métodos relacionados con los casos de estudio: su objetivo primario, los tipos de datos primarios en los que se basan, y si el diseño es fijo o flexible.

Método	**Objetivo primario**	**Datos primarios**	**Diseño**
Survey	*Descriptivo* de una situación o fenómeno	Cuantitativo	Fijo
Caso de estudio	*Exploratorio*, para descubrir lo que sucede, generando ideas e hipótesis para nuevas investigaciones	Cualitativo	Flexible
Experimento	*Explicativo*, buscando una explicación de una situación o problema	Cuantitativo	Fijo
Investigación en acción	*Mejora*, intentando mejorar algún aspecto del fenómeno estudiado	Cualitativo	Flexible

Tabla 1 Características generales de métodos de investigación relacionados con casos de estudio

En resumen, las características clave de una investigación basada en casos de estudio son las siguientes: (1) es flexible, de manera que se pueden gestionar las características complejas y dinámicas de los fenómenos del mundo real, como se dan en la ingeniería del software; (2) su conclusiones se basan en una "cadena de evidencia" clara, ya sea cualitativa o cuantitativa, establecida de forma planificada y consistente a partir de múltiples fuentes; y (3) mejora el conocimiento existente, pues está basada en una teoría previamente establecida, si existe, o en una nueva teoría, en caso contrario.

El proceso de investigación basada en casos de estudio se puede resumir en cinco pasos:

1. *Diseño del caso de estudio*: se definen los objetivos y el ámbito del caso de estudio y se realiza una planificación, todo lo cual se recoge en el protocolo del caso de estudio.
2. *Preparación para la recogida de datos*: se definen procedimientos y protocolos para la recogida de datos.
3. *Recolección de la evidencia*: se ejecuta el caso de estudio y se recogen datos (por ejemplo, mediante entrevistas, observación y uso de datos de archivo).
4. *Análisis de los datos recogidos*: se genera y analiza nuevo conocimiento, por ejemplo a través de la búsqueda de patrones en los datos recogidos. Se debe mantener una cadena de evidencia desde los resultados a los datos originales.
5. *Creación de un informe*: se deben incluir suficientes datos y ejemplos para que el lector comprenda la cadena de evidencia mencionada en el paso anterior.

El proceso anterior es un proceso similar al de cualquier otro estudio empírico. Una característica fundamental del proceso anterior es que se lleva a cabo iterativamente, y la recogida y análisis de datos se pueden realizar de forma incremental. Por ejemplo, si no se han recogido datos suficientes para el análisis, se pueden recoger más datos. Existe un límite en cuanto a la flexibilidad: los objetivos se deben haber establecido desde el principio. Si los objetivos cambian, no se trata de un cambio en el caso de estudio, sino de uno nuevo.

Para validar las propuestas recogidas en esta tesis doctoral se hace uso de dos casos de estudio: (1) reutilización de requisitos en dominios verticales de aplicación, dentro de tres proyectos desarrollados por pequeñas y medianas empresas de software en el marco del proyecto GARTIC (Sección 1.4.3.1). Estos proyectos tratan con el desarrollo de aplicaciones de comercio electrónico, con un aplicación genérica para terminales punto de venta y con un gestor de contenidos (ver Sección 2.7.3); y (2) los sistemas teleoperados para mantenimiento de cascos de buques (ver Sección 3.2), que son una familia de sistemas robóticos desarrollados en la Universidad Politécnica de Cartagena por la DSIE (División de Sistemas e Ingeniería Electrónica) en el marco del proyecto europeo EFTCoR (*Environmental Friendly and Cost-Effective Technology for Coating Removal,* V Programa Marco [93]). El proyecto –en el que han intervenido dos grupos de investigación y ocho empresas europeas– ha abordado mediante una nueva familia de sistemas robóticos un problema crítico para la industria naval europea: la preparación de las superficies del casco para su pintado de manera respetuosa con el medio ambiente.

1.3.2 Investigación en acción

Investigación en acción (I-A) [34] es un método de investigación cualitativo (ver Tabla 1) en el que los investigadores trabajan conjuntamente con los expertos en el dominio para mejorar la comprensión de un fenómeno determinado. I-A ha sido aplicada tradicionalmente en ciencias sociales y recientemente ha recibido la atención del campo de Sistemas de Información primero y de Ingeniería del Software después, dado que se trata de una aproximación a la investigación que puede ser aplicada en el estudio los efectos de cambios en los métodos de desarrollo y mantenimiento sistemas y/o de software [35], como en el caso de esta tesis doctoral. En este sentido Baskerville [34] afirma que los procesos sociales complejos, como los que vienen dados por el uso de tecnologías de la información en las organizaciones, pueden ser estudiados adecuadamente siguiendo una aproximación de I-A, introduciendo cambios en tales procesos y observando los efectos de dichos cambios.

I-A promueve un proceso de aprendizaje reflexivo y una búsqueda de soluciones prácticas que involucran tanto a los investigadores como a los profesionales. La aplicación de I-A se realiza mediante un proceso cíclico en el cual participan todas las partes involucradas en la investigación, que examinan la situación actual con la intención de mejorarla en algún sentido [294]. I-A no se refiere a un método de investigación concreto, sino más bien a una clase de métodos de investigación que, de acuerdo con Baskerville [34], comparten las siguientes características:

- Enfoque sobre un problema práctico.
- Orientación hacia la acción y el cambio.
- Colaboración entre los participantes.
- Un modelo de procesos que involucra una serie de pasos sistemáticos, en ocasiones iterativos.

En una aplicación de I-A están implicados los siguientes roles:

- El *investigador* (*the researcher*), aquel que investiga.
- El *objeto investigado* (*the researched*) en el proceso de investigación, que debe ser mejorado en algún sentido.
- El grupo crítico de referencia (GCR, *critical reference group*), es decir, aquel para el que se investiga, pues tiene un problema que necesita ser resuelto. Según I-A, el GCR ha de participar también en el proceso de investigación, aunque pueda hacerlo menos activamente que el investigador.
- Los beneficiarios (*stakeholders*), que son todas aquellas organizaciones que pueden beneficiarse de los resultados de la investigación.

I-A se puede aplicar en distintas modalidades. En esta tesis doctoral, I-A se ha aplicado en su variante *participativa* [294], en la cual el GCR pone en práctica las recomendaciones propuestas por el investigador, con el cual comparte los efectos y resultados. Mediante el uso de cuestionarios se ha tratado de cuantificar el valor que ha tenido para el GCR la propuesta bajo estudio, si bien la aproximación se ha de seguir considerando cualitativa. Una variante *empírica* de I-A habría sido difícil de aplicar en

esta tesis doctoral porque hubiera requerido del GCR la realización de un amplio y sistemático registro de acciones y efectos, y la existencia de registros previos con los cuales comparar y establecer las posibles mejoras.

Las actividades cíclicas que se realizan en I-A se describen en la Figura 1. El ciclo comienza con (1) la actividad de *planificación*, en la que se identifican las cuestiones que guían la investigación y las acciones que pueden ayudar a resolver tales cuestiones; (2) se continúa con una actividad de *acción*, en la cual a partir de los resultados de la planificación el investigador introduce una variación de la práctica cuidadosa, deliberada y controlada; (3) se prosigue con una actividad de *observación* o *evaluación*, en la cual se recoge información sobre los efectos de la acción, y unas primeras lecciones aprendidas comienzan a tomar forma; y (4) cada ciclo termina con una actividad de *reflexión*, en la cual los resultados se comparten entre todos los interesados: ciertas soluciones se pueden considerar suficientemente maduras, al tiempo que se pueden refinar nuevas cuestiones susceptibles de ser analizadas en un nuevo ciclo de I-A.

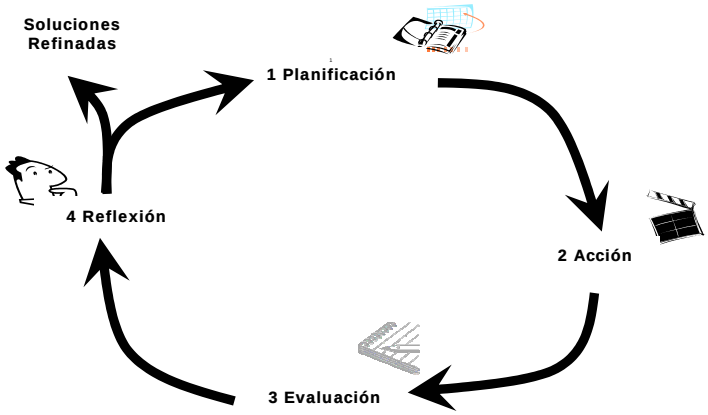

Figura 1 Proceso de aplicación de I-A

En esta tesis doctoral I-A se ha aplicado en el ámbito del caso de estudio de los sistemas teleoperados para mantenimiento de cascos de buques, referenciado por primera vez en la sección anterior.

1.3.3 Revisiones sistemáticas de la literatura

Una interesante herramienta conceptual para estudiar el estado del arte en un campo de conocimiento son las revisiones sistemáticas de la literatura (RSL), que recientemente Kitchenham [163] ha adaptado a la Ingeniería del Software a partir de su uso en disciplinas científicas más maduras como la Medicina. Es obvio que cualquier investigación se ha de iniciar con un estudio del estado del arte. Sin embargo, si dicho estudio no se realiza de forma sistemática, su valor científico puede quedar en entredicho. En todo este apartado nos basamos en el trabajo de Kitchenham [163].

Una RSL se acomete con el objetivo general de sintetizar la evidencia que existe en la literatura en relación con alguna cuestión sobre, por ejemplo, un método, un procedimiento, una técnica, cierta tecnología o una herramienta. Mediante una RSL, en un campo de conocimiento determinado, se puede diseñar un marco en el que situar nuevas actividades de investigación en dicho campo, o se puede identificar un área de la investigación actual en dicho campo en la que se requiere más investigación. Una RSL

también se puede utilizar para examinar en qué medida la evidencia empírica soporta (o contradice) determinadas hipótesis, o incluso para ayudar en la generación de nuevas hipótesis. Si se revisan aproximaciones cuantitativas, utilizando técnicas de metaanálisis se pueden combinar los resultados individuales para llegar a resultados que los estudios individuales, de grano más fino, no pueden predecir. Por contra, uno de los principales problemas que se suelen asociar con las revisiones sistemáticas es que su realización requiere mucho más esfuerzo que las revisiones de la literatura tradicionales.

Una RSL se realiza en tres etapas principales: *planificación*, *ejecución* e *informe* (ver Tabla 2). Un aspecto fundamental es que la RSL se debe realizar siguiendo un protocolo predefinido que garantice que tanto los trabajos que avalan la hipótesis de partida como aquellos que no lo hacen puedan ser estudiados. Con un protocolo predeterminado se trata de evitar eventuales sesgos en los resultados de la RSL (si bien, obviamente, no se pueden evitar posibles sesgos en la publicación de los estudios primarios que se analizan en la RSL). En este sentido, por ejemplo en Medicina, son dos investigadores los que evalúan el protocolo de revisión. La estrategia de búsqueda de una RSL se documenta de manera que los lectores puedan valorar su rigor y completitud y para que el proceso sea repetible (teniendo en cuenta que es casi imposible replicar las búsquedas en bibliotecas digitales).

1. Planificación (*planning the review*)
1.1. Identificación de la necesidad de la revisión. 1.2. Definición de las condiciones de encargo de la revisión (*commissioning the review*), si por algún motivo debe ser llevada a cabo por un equipo de investigadores externo. 1.3. Definición de la(s) pregunta(s) de la revisión. 1.4. Desarrollo del protocolo de la revisión. 1.5. Evaluación del protocolo de la revisión.
2. Ejecución (*conducting the review*)
2.1. Identificación de los estudios relevantes sobre el tema. 2.2. Selección de estudios primarios. 2.3. Evaluación de la calidad de los estudios primarios. 2.4. Extracción de datos de los estudios primarios. 2.5. Síntesis de datos.
3. Informe (*reporting the review*)
3.1. Especificación de la diseminación de datos. 3.2. Formateo del informe principal. 3.3. Evaluación del informe.

Tabla 2 Proceso de aplicación de una RSL [163]

En resumen, el protocolo de la revisión especifica formalmente (1) el ámbito del problema; (2) las fuentes de información; (3) las cadenas de búsqueda, que tratan de encontrar tanta literatura relevante como sea posible (por lo general la búsqueda se realiza automáticamente); (4) los criterios para la inclusión y la exclusión de los trabajos encontrados en las búsquedas; (5) el análisis cuantitativo que debe ser realizado (en caso de que fuera necesario); y (6) las plantillas para recopilar los datos recogidos de los trabajos, incluyendo los criterios de calidad mediante los cuales se evalúa cada estudio primario. Kitchenham [163] propone un protocolo de la revisión que se reproduce en la Tabla 3.

El grupo del profesor Travassos extiende este protocolo en una versión más elaborada (Biolchini *et al.* [42]). Para terminar esta breve introducción sobre las RSL se puede destacar que Kitchenham *et al.* presentan recientemente una RSL sobre el uso de revisiones sistemáticas en Ingeniería del Software [164].

1. Antecedentes de la RSL. 2. Cuestiones de la investigación. 3. Estrategia de búsqueda de los estudios primarios, incluyendo los términos de búsqueda y las fuentes en las que realizar la búsqueda –bibliotecas digitales, revistas, actas de congresos–. 4. Criterios de selección de estudios: qué estudios son incluidos y excluidos de la RSL. 5. Procedimientos de selección, que indican cómo se aplican los criterios de selección: cuántas personas van a evaluar cada estudio primario, y cómo se resolverán los desacuerdos entre revisores. 6. Listas de chequeo y procedimientos de valoración de la calidad de los estudios. 7. Estrategias de extracción de datos. 8. Síntesis de los datos extraídos. Si es necesario se establece un metaanálisis formal. 9. Estrategia de diseminación de los resultados. 10. Planificación del proyecto de la RSL.

Tabla 3 Protocolo de una RSL [163]

En esta tesis doctoral las RSL se han utilizado para estudiar el estado del arte de diversos campos de conocimiento: las amenazas y riesgos presentes en el desarrollo global de software (Sección 2.10.2) y la generación de requisitos textuales a partir de modelos de ingeniería del software (Sección 4.2).

1.4 Marco de realización de la tesis

1.4.1 Grupo de investigación y trabajos previos

El autor de esta tesis doctoral forma parte del Grupo de Investigación en Ingeniería del Software de la Universidad de Murcia (GIS, grupo E097-01, www.um.es/giisw), desde febrero de 1995, fecha en que se fue contratado como becario del Proyecto PASO OOAP. Posteriormente, en octubre de 1996 se incorporó al Departamento de Informática y Sistemas como ayudante de escuela universitaria en el área de Lenguajes y Sistemas Informáticos y desde julio de 2002 desempeña labores de titular de escuela universitaria en dicha área.

La tesis doctoral que se describe en esta memoria se encuadra en una línea de investigación en reutilización de requisitos, que se inicia en 2000 en el GIS con la hipótesis de partida de que con la introducción de un método de reutilización de requisitos se puede mejorar el proceso de IR y por tanto la calidad del producto final y la productividad del desarrollo. Para validar esta hipótesis se define un método de reutilización de requisitos, denominado SIREN (*SImple REuse of RequiremeNts*). Con posterioridad, esta línea de trabajo se refina en el marco de esta tesis doctoral en las siguientes tres líneas de investigación:

1. IR y reutilización (métodos SIREN y SIRENgsd, desarrollada en el Capítulo 2).
2. IR para líneas de productos (método SIRENspl, Capítulo 3).
3. Generación de especificaciones de requisitos textuales a partir de modelos de ingeniería del software del sistema (en el ámbito de SIRENspl, Capítulo 4).

En nuestra opinión se trata de tres líneas diferenciadas pero complementarias, que se han ido construyendo sucesivamente, cada una sobre la anterior, como se resume en la Sección 1.4.2. Con anterioridad al inicio de la línea de investigación con SIREN, en el GIS se habían realizado trabajos en el ámbito de la especificación de requisitos, modelado conceptual, métodos formales y prototipado. La reutilización de requisitos, sin embargo, no se había abordado expresamente. En particular, en cuanto a la labor investigadora de este doctorando, los resultados más destacados de esta etapa anterior fueron los siguientes:

- En [218] se proponía la integración del prototipado rápido o desechable (basado en un lenguaje de especificación formal orientado a objetos denominado OASIS, desarrollado en la Universidad Politécnica de Valencia) y del prototipado evolutivo (basado en cualquier lenguaje de programación comercial que soportara el diseño por contrato) para obtener la especificación validada de un sistema de información, que automáticamente diera lugar a un primer prototipo evolutivo de la aplicación software. Se desarrollaron prototipos de herramientas para soportar la animación del prototipo desechable formal y la generación del primer prototipo evolutivo.
- En [230] se realizaba la aplicación de OBJ3, un lenguaje de especificación algebraico basado en reescritura de términos, en la descripción, validación y verificación formales del software de control de un microscopio efecto túnel desarrollado en el Departamento de Física de la Universidad de Murcia.
- En [113] se presentaba una propuesta para el modelado del negocio basada en UML a partir de la cual se podían obtener los primeros modelos del análisis de requisitos del software, en concreto un modelo de casos de uso y un modelo conceptual.
- En [112] se establecía una correspondencia entre la arquitectura de tres modelos del método OOram y un nuevo proceso de desarrollo orientado a objetos basado en un conjunto de técnicas de UML, que incluía el modelado del negocio descrito en [113].

Como se ha comentado, estos trabajos no abordan directamente la reutilización de requisitos, sino que se dirigen al modelado y prototipado de los requisitos de un sistema, utilizando tanto técnicas formales como industriales. No obstante, esta experiencia previa en investigación ha servido como base para abordar la realización de esta tesis doctoral.

1.4.2 Proyectos de investigación

La presente tesis doctoral se ha desarrollado en relación con los siguientes proyectos de investigación, desarrollo e innovación, a través de los cuales este doctorando ha mantenido relaciones con otros grupos de investigación.

1.4.2.1 SIRENrm

Con el proyecto SIRENrm (*SImple REuse of software requiremeNts and rigorous modelling*, ver resumen en la Tabla 4) se inicia la línea de investigación sobre IR y reutilización en el marco de la cual comienza esta tesis doctoral. SIREN constituye un

subproyecto de un proyecto coordinado mayor, DOLMEN (*Distributed Objects, Languages, Models and Environments*), cuyo investigador principal es el Dr. Isidro Ramos Salavert de la Universidad Politécnica de Valencia. A través de la realización de este proyecto SIREN se realiza una primera definición del método homónimo, que se aplica a las definiciones iniciales de dos catálogos de requisitos reutilizables: seguridad y protección de datos personales. En el marco de SIREN se realiza además la primera versión de un prototipo de la herramienta de soporte de SIREN, SirenTool.

Título	SIRENrm: *SImple Reuse of software requiremeNts and rigorous modelling*
Entidad financiadora	Comisión Interministerial de Ciencia y Tecnología (CICYT), TIC2000-1673-C06-02
Importe de la ayuda	89.190 € (subproyecto SIRENrm)
Entidades participantes	Universidad Politécnica de Valencia, Universidad de Murcia, Universidad de Sevilla, Universidad de Granada, Universidad de Valladolid y Universidad de Castilla-La Mancha
Investigador principal	Dr. José Ambrosio Toval Álvarez
Duración:	Enero 2001-Diciembre 2003
Número de investigadores participantes	8 (subproyecto SIRENrm)

Tabla 4 Resumen del proyecto SIRENrm

1.4.2.2 PRESSURE

PRESSURE (*PREcise Software modelS and reqUirements ReusE*, ver Tabla 5) es un subproyecto del proyecto coordinado DYNAMICA (*DYNamic and Aspect-Oriented Modeling for Integrated Component-based Architectures*), cuyo investigador principal es el Dr. Isidro Ramos Salavert. Los objetivos iniciales del proyecto PRESSURE en la línea de investigación de reutilización de requisitos continúan los de SIRENrm. Así, continuando la línea de investigación de IR y reutilización, durante este proyecto se refinan los catálogos de seguridad y protección de datos personales creados en el proyecto anterior, y a partir de la experiencia conseguida se reflexiona sobre los aspectos clave en la reutilización de requisitos. PRESSURE sirve también para establecer una colaboración con la Universidad Politécnica de Cartagena en el ámbito de los sistemas teleoperados para mantenimiento de cascos de buques, con el objetivo de desarrollar un catálogo de requisitos reutilizables para dicho dominio de aplicación vertical. Esta colaboración se convierte en el germen de la segunda línea de investigación de esta tesis, IR para líneas de productos.

1.4.2.3 DÉDALO

DÉDALO (*Desarrollo de Sistemas de Calidad Basado en Modelos y Requisitos*, ver Tabla 6) es un subproyecto del proyecto coordinado META (*Models, Environments, Transformations and Applications*), cuyo investigador principal es de nuevo el Dr. Isidro Ramos Salavert. En este proyecto coordinado se continúa con la relación con la Universidad Politécnica de Cartagena, a través del caso de estudio de los sistemas teleoperados para mantenimiento de cascos de buques, y se inicia una tercera nueva línea de investigación en cuando a la generación de especificaciones de requisitos a partir de modelos en ingeniería del software, que consideramos puede complementar el trabajo realizado en la línea de IR para líneas de productos.

Título	PRESSURE: *PREcise Software modelS and reqUirements ReusE*
Entidad financiadora	Comisión Interministerial de Ciencia y Tecnología (CICYT), TIC2003-07804-C05-05
Importe de la ayuda	163.280 € (Subproyecto PRESSURE)
Entidades participantes	Universidad Politécnica de Valencia, Universidad Politécnica de Cartagena, Universidad de Castilla La Mancha, Universidad Carlos III, Universidad de Murcia
Investigador principal	Dr. José Ambrosio Toval Álvarez
Duración:	Enero 2004-Noviembre 2006
Número de investigadores participantes	10 (Subproyecto SIRENrm)

Tabla 5 Resumen del proyecto PRESSURE

Título	DÉDALO: *Desarrollo de Sistemas de Calidad Basado en Modelos y Requisitos*
Entidad financiadora	Comisión Interministerial de Ciencia y Tecnología (CICYT), TIN2006-15175-C05-03
Importe de la ayuda	110.000€ (Subproyecto DÉDALO)
Entidades participantes	Universidades de Murcia, Valencia, Cartagena, Castilla-La Mancha, ESI (*European Software Institute*) y varias EPO
Investigador principal	Dr. José Ambrosio Toval Álvarez
Duración:	Enero 2007-Diciembre 2009
Número de investigadores participantes	8 (Subproyecto DÉDALO)

Tabla 6 Resumen del proyecto DÉDALO

1.4.2.4 GREIS

GREIS (*Global Requirements Engineering for Information Systems*, Tabla 7) forma parte del proyecto coordinado MELISA (*Metodología para el desarrollo global del software*), cuya investigadora responsable es la Dra. Aurora Vizcaíno Barceló de la Universidad de Castilla-La Mancha. En la línea de IR y reutilización, en este proyecto se han comenzado a abordar los desafíos de la IR en el desarrollo global de software, estudiando la adaptación de SIREN y de los catálogos de requisitos al desarrollo global de software (dando lugar a la definición de una extensión de SIREN para el desarrollo global de software denominada SIRENgsd).

Título	GREIS: *Global Requirements Engineering for Information Systems*
Entidad financiadora	Consejería de Ciencia y Tecnología. Junta de Comunidades de Castilla-La Mancha, PAC08-0142-335
Importe de la ayuda	19.040€ (nodo Universidad de Murcia)
Entidades participantes	Universidad de Castilla-La Mancha y Universidad de Murcia
Investigador principal	Dr. José Ambrosio Toval Álvarez
Duración:	Enero 2008-Diciembre 2010
Número de investigadores participantes	6 (nodo Universidad de Murcia)

Tabla 7 Resumen del proyecto GREIS

1.4.2.5 PANGEA

PANGEA (*Process for globAl requiremeNts enGinEering and quAlity*, Tabla 8) es un subproyecto del proyecto PEGASO (*Procesos para la mEjora del desarrollo GlobAl del SOftware*). Este proyecto coordinado, cuyo investigador principal es el Dr. Mario Piattini Velthuis de la Universidad de Castilla-La Mancha, ha sido recientemente aprobado por la CICYT. Continuando la línea de IR y reutilización en el desarrollo global de software, iniciada en el proyecto GREIS, el objetivo principal de PANGEA consiste en la definición de un método homónimo de IR que pueda ser usado en el contexto del desarrollo global de software. En este subproyecto se trata de (1) compilar los problemas, desafíos y riesgos que amenazan la IR en un ambiente distribuido de desarrollo, compilar las soluciones propuestas en la literatura para gestionarlos, y crear un repositorio de buenas prácticas, en el que se clasifiquen las amenazas que afectan a la IR distribuida, y se propongan guías o salvaguardas que ayuden a manejarlas; (2) definir un proceso para la IR global, denominado PANGEA, que materialice la implantación de las guías o salvaguardas generales propuestas en el repositorio de buenas prácticas. En la definición de este método se habrá de tener en cuenta especialmente la reutilización y negociación de los requisitos y la calidad de los mismos; (3) definir un modelo de referencia de requisitos para el desarrollo global de software, con vistas a establecer formalmente los elementos que forman parte de la especificación de requisitos y el modelo de trazabilidad entre requisitos y entre requisitos y el resto de artefactos software, que sirva como base para intercambiar la especificación de requisitos desarrollada en cada ubicación, y que permita establecer los modelos de requisitos a utilizar en cada localización como vistas de este modelo de referencia; (4) establecer un proceso de gestión de los requisitos de seguridad en el ámbito de PANGEA; (5) definir técnicas de evaluación de la calidad de modelos de requisitos para el desarrollo global de software; (6) desarrollar herramientas de soporte; y (7) validar empíricamente el proceso y las técnicas.

Título	PANGEA: *Process for globAl requiremeNts enGinEering and quAlity*
Entidad financiadora	Comisión Interministerial de Ciencia y Tecnología (CICYT)
Importe de la ayuda	191.000€ (nodo Universidad de Murcia)
Entidades participantes	Universidad de Castilla-La Mancha, Universidad de Murcia y tres EPO
Investigador principal	Dr. José Ambrosio Toval Álvarez
Duración:	Enero 2010- Diciembre 2012
Número de investigadores participantes	11 (Subproyecto PANGEA)

Tabla 8 Resumen del proyecto PANGEA

1.4.3 Proyectos de transferencia tecnológica

La presente tesis doctoral se ha desarrollado en relación con el siguiente proyecto de transferencia tecnológica, a través del cual este doctorando ha mantenido relaciones con el mundo empresarial.

1.4.3.1 GARTIC

En relación con la línea de investigación de IR y reutilización, GARTIC (*Gestión Automatizada de Requisitos basada en reutilización para PYMES del sector TIC*, Tabla 9) es un proyecto de innovación centrado en la transferencia tecnológica de un proceso general de IR y del método SIREN de reutilización de requisitos a cinco empresas de desarrollo de software en la Región de Murcia. El objetivo general de GARTIC consiste en la introducción en las empresas participantes de un método de IR basado en reutilización (técnicas, proceso, guías y herramienta CARE, *Computer-Aided Requirements Engineering*), que ayude a la mejora de los índices de calidad y productividad en el desarrollo de software correcto en pequeñas y medianas empresas de desarrollo de software. SIREN ha sido puesto en práctica en cinco proyectos piloto en el marco de GARTIC –uno por empresa– y redefinido de acuerdo con la retroalimentación recibida en todos ellos. GARTIC también ha servido para redefinir SIREN y para desarrollar una nueva versión de SirenTool.

Título	GARTIC: *Gestión Automatizada de Requisitos basada en reutilización para PYMES del sector TIC*
Entidad financiadora	Consejería de Industria y Medio Ambiente de la Comunidad Autónoma de la Región de Murcia
Importe de la ayuda	32.480€ (nodo Universidad de Murcia)
Entidades participantes	CenTIC (*Centro Tecnológico de las Tecnologías de la Información y las Comunicaciones de la Región de Murcia*); GIS; Empresas: Adalid Management y Outsourcing S.L., Bahia Information Technology S.A., Foro Digital S.L., Global Touch Express S.L., S.Q.A. MURCIA S.L.
Investigador principal	Dr. José Ambrosio Toval Álvarez
Duración:	Enero 2008-Febrero 2009
Número de investigadores participantes	6 (nodo Universidad de Murcia)

Tabla 9 Resumen del proyecto GARTIC

1.5 Organización de la memoria

La organización de esta memoria de tesis doctoral sigue las tres líneas de investigación presentadas en la Sección 1.4.

En el Capítulo 2 se estudia en profundidad la línea más general, la de IR y reutilización. El contenido de este capítulo se puede resumir de la siguiente manera: (1) para establecer un contexto adecuado, primero se presenta la disciplina de IR en general y después el estado del arte en reutilización de requisitos; (2) acto seguido se presenta en detalle el método SIREN (modelo de procesos, técnicas, guías, herramienta), así como los casos de estudio en los que se ha aplicado SIREN, y un conjunto de lecciones aprendidas en forma de ocho cuestiones clave para la reutilización de requisitos; (3) finalmente se presenta un catálogo de riesgos y salvaguardas en el desarrollo global de software, y una extensión de SIREN, que denominamos SIRENgsd, para tratar con el desarrollo global de software, que tiene en cuenta los riesgos y salvaguardas incluidos en dicho catálogo.

En el Capítulo 3 se trata la línea de IR para líneas de productos, mostrándose en detalle la propuesta que extiende SIREN para el análisis de líneas de productos software. En resumen, (1) se comienza repasando el estado del arte en modelos de variabilidad generales o de requisitos; (2) seguidamente se presenta SIRENspl, una propuesta diseñada expresamente para un dominio vertical concreto, como es el caso de estudio de los sistemas teleoperados para mantenimiento de cascos de buques, y se describe en profundidad el desarrollo de dicho caso de estudio y las lecciones aprendidas; (3) finalmente se muestra una visión general del prototipo de una herramienta de soporte para SIRENspl, desarrollado con Eclipse EMF/GMF.

Construyendo sobre lo anterior, en el Capítulo 4 se detalla la línea de investigación que estudia el uso combinado de requisitos y de modelos en ingeniería del software. En este capítulo, (1) se comienza reflejando el estado del arte en generación de requisitos textuales (preferentemente en lenguaje natural pero también en lenguaje formal) partiendo de modelos de ingeniería del software (preferentemente modelos gráficos); (2) a continuación se muestra una propuesta que se integra en SIRENspl y que liga los modelos de ingeniería del software definidos en el Capítulo 3 con los requisitos textuales definidos en el Capítulo 2; (3) finalmente se muestra una implementación de dicha propuesta, a través del estándar del OMG QVT, y se comenta su aplicación retrospectiva al caso de estudio presentado en el Capítulo 3.

En el Capítulo 5 se presentan las conclusiones y vías futuras generales de la tesis doctoral, aunque unas conclusiones parciales más detalladas se incluyen al final de cada capítulo 2, 3 y 4. Para terminar, en este Capítulo 5 se repasan los resultados obtenidos en el desarrollo de esta tesis doctoral.

Finalmente, el Capítulo 6 recoge la bibliografía y referencias.

2 SIREN: Un método práctico de ingeniería de requisitos basado en reutilización

2.1 Introducción

Este capítulo está dedicado a la línea de investigación de Ingeniería de Requisitos (IR) y reutilización, cuyo resultado más destacado es la definición del método SIREN (*SImple REuse of RequiremenNts*) [226, 280, 281, 283, 284], una propuesta de reutilización de requisitos desarrollada por el Grupo de Investigación en Ingeniería del Software de la Universidad de Murcia (GIS).

En este Capítulo 2 primero se proporciona una visión general de la IR (Sección 2.2) y del estado de la investigación en reutilización de requisitos (Sección 2.3). En la Sección 2.4 comienza la descripción del método SIREN, concebido como una extensión sencilla y práctica para cualquier método de IR en el que se desee incorporar reutilización (a modo de ejemplo, en la Sección 2.4.1 se muestra brevemente la conexión con Métrica 3). El modelo de procesos revisado de SIREN se describe en la Sección 2.5. Unas guías de aplicación de SIREN se muestran en la Sección 2.6. Un modelo inicial de referencia de requisitos se puede encontrar en la Sección 2.6.9 (modelo inicial porque muestra de forma muy básica los elementos involucrados en la descripción de SIREN: un modelo refinado se encuentra en el Capítulo 4). En la Sección 2.7 se recogen experiencias de aplicación de SIREN: en la Sección 2.7.1 se describen los catálogos de requisitos de seguridad, en la Sección 2.7.2 una experiencia inicial con la auditoría informática usando los catálogos anteriores y en la Sección 2.7.3 una serie de tres casos de estudio en pequeñas y medianas empresas. En la Sección 2.8 se listan ocho cuestiones clave sencillas que, a la luz de las experiencias anteriores, en nuestra opinión debería tener cualquier método basado en reutilización. En la Sección 2.9 se hace referencia a una herramienta, SirenTool, que da soporte a SIREN y a dichas ocho cuestiones clave. En la Sección 2.10 se trata el desarrollo global de software, presentando una revisión sistemática sobre los riesgos y salvaguardas que se plantean en el desarrollo global de software para la IR (Sección 2.10.2), mostrando cómo se pueden estructurar tales riesgos y salvaguardas en un catálogo (Sección 2.10.3), y finalmente describiendo SIRENgsd, una extensión de SIREN para abordar la IR en un contexto de desarrollo global (Sección 2.10.4). Finalmente, en la Sección 2.11 se muestran las conclusiones relacionadas con la línea de trabajo de reutilización y requisitos.

2.2 Ingeniería de requisitos

2.2.1 Introducción

Siguiendo la primera parte de Nicolás y Toval [227, 228], en esta sección se extiende la breve introducción que sobre la IR se ha realizado en la Sección 1.1.1, presentando una visión esquemática sobre cuestiones generales de gestión de requisitos e incluyendo técnicas que son de utilidad para cualquier organización que desee mejorar la madurez de su proceso de desarrollo.

El resto de esta sección se estructura de la siguiente manera. En la Sección 2.2.2 se trata la mejora del proceso de IR en el marco de uno de los estándares de evaluación del proceso más conocidos, CMMI. En la Sección 2.2.3 se describe una selección de

técnicas clásicas de IR, que pueden ayudar a una organización que decida mejorar la capacidad de su proceso de IR. En la Sección 2.2.4 se mencionan enfoques alternativos de IR. Finalmente la Sección 2.2.5 se dedica al soporte automatizado para la IR, las herramientas CARE (*Computer-Aided Requirements Engineering*).

2.2.2 Ingeniería de requisitos y CMMI

La industria en general, y la relacionada con las TIC en particular, está cada vez más interesada en la adopción de estándares, lo cual en nuestra opinión es debido fundamentalmente a dos cuestiones: (1) el interés en la obtención de certificados de calidad (como por ejemplo ISO 9001 [149]), con los que enfrentarse a las exigencias, cada vez mayores, de los clientes y de las administraciones públicas, especialmente en el marco de un desarrollo de software que cada vez se encuentra más distribuido y globalizado; (2) la búsqueda de los beneficios de la cultura impuesta por las propuestas de evaluación y mejora del proceso software, que dentro de los estándares de ingeniería del software, han sido concebidas para determinar y mejorar la capacidad de los procesos en las organizaciones, al objeto de que éstas desarrollen productos de calidad de manera consistente y predecible. Entre los principales estándares de evaluación y mejora del proceso software encontramos ISO/IEC 15504 (SPICE) [147] y CMMI (*Capability Maturity Model Integration*) [58, 59], que probablemente es el más difundido. En este apartado se resume el tratamiento que CMMI hace de la gestión de los requisitos.

CMMI relaciona una serie de principios y prácticas con la madurez de los procesos en una organización, con el objetivo de introducir progresivamente mejoras en tales procesos. A diferencia de los modelos CMM que lo preceden, CMMI integra las disciplinas de ingeniería de software e ingeniería de sistemas en un único marco de referencia, eliminando así redundancias e inconsistencias presentes en los anteriores modelos. Aunque los requisitos afectan muchas áreas de proceso, CMMI define dos áreas expresamente relacionadas con los requisitos: Gestión de Requisitos (GR, *Requirements Management*) en el Nivel 2 de la representación escalonada y Desarrollo de Requisitos (DR, *Requirements Development*) en el Nivel 3. Para implementar DR es preciso haber institucionalizado GR. Además los requisitos tienen relación con otras áreas de proceso de CMMI, como Planificación del Proyecto, Gestión de Acuerdos con los Proveedores, Gestión de la Configuración, Validación, Verificación y Gestión del Riesgo.

El objetivo fundamental de GR es la gestión de los requisitos del proyecto y la identificación de inconsistencias de los requisitos con los planes del proyecto y con los productos de trabajo del proyecto. En otras palabras, se trata de obtener un acuerdo sobre el conjunto de requisitos que serán implementados y mantener actualizado dicho conjunto de requisitos a lo largo de todo el proyecto. Las metas especificas de esta área de proceso son las de obtener y analizar los requisitos (que CMMI denomina en nuestra opinión de forma un tanto ambigua *Obtain an Understanding of Requirements*), obtener la aprobación de éstos, gestionar los cambios de los requisitos, mantener la trazabilidad bidireccional de los requisitos entre sus fuentes y los artefactos que se derivan de ellos e identificar inconsistencias entre el desarrollo y los requisitos.

El objetivo fundamental de DR es analizar los requisitos del cliente y extraer a partir de ellos los requisitos de la aplicación o producto que se ha de desarrollar. En esta área de

proceso se incluye el análisis de los requisitos de los clientes, construyendo un modelo del dominio del problema y relacionando los requisitos de los clientes con dicho modelo, para seleccionar y desarrollar los requisitos que han de ser soportados por el producto o aplicación. Después se construye una especificación de requisitos, es decir, una especificación del comportamiento externo del sistema que permita la comprensión de las funciones del mismo, que incluya los requisitos de interfaz, y que permita la validación de los requisitos.

Para mejorar la evaluación y mejora de los procesos de IR, Sawyer *et al.* [267] proponen un modelo de madurez de capacidad específicamente centrado en la IR, debido a que en su opinión la IR no está suficientemente cubierta en los estándares generales de ingeniería del software. El modelo REPM (*Requirements Engineering Process Maturity*) define un conjunto de buenas prácticas en IR, y ordena las organizaciones en un sencillo esquema de tres niveles: (1) *Inicial*; (2) *Repetible*; y (3) *Definido*. Las organizaciones del Nivel 1 no tienen un proceso de IR definido, mientras que las organizaciones de Nivel 2 tienen definidos estándares para documentos de requisitos y descripciones de requisitos, y han introducido políticas y procedimientos para gestión de requisitos. Estos dos niveles se corresponden con los niveles 1 y 2 de la representación escalonada de CMMI. Finalmente, el Nivel 3 de REPM se asigna a las organizaciones que tienen un proceso de IR definido y basado en buenas prácticas y técnicas, con un activo proceso de mejora del proceso, y que evalúan nuevas técnicas y herramientas de forma objetiva. Este nivel se corresponde con los niveles 3 al 5 de CMMI. Además REPM se basa en la adopción incremental de una Guía de buenas prácticas de IR, que contiene un modelo de procesos en espiral y diez guías de buenas prácticas (ver Tabla 10).

1.	Definir una estructura de documento estándar.
2.	Hacer el documento fácil de cambiar.
3.	Identificar cada requisito de forma única.
4.	Definir políticas para la gestión de requisitos.
5.	Definir plantillas estándar para la descripción de requisitos.
6.	Utilizar un lenguaje simple, consistente y conciso.
7.	Organizar inspecciones formales de requisitos.
8.	Definir listas de validación.
9.	Utilizar listas de chequeo para el análisis de los requisitos.
10.	Planificar la resolución de conflictos.

Tabla 10 Guía de buenas prácticas de REPM

2.2.3 Conceptos y técnicas generales de ingeniería de requisitos

Una descripción completa de los conceptos, técnicas, procesos y herramientas que soportan las áreas de proceso de CMMI relacionadas con la IR queda fuera del ámbito de esta memoria de tesis doctoral. En este apartado presentamos conceptos fundamentales junto con una selección de técnicas utilizadas con carácter general en IR, que ayudan a implementar buenas prácticas mencionadas por CMMI en relación con esta disciplina, y que tienen más relación con la línea de investigación de IR y reutilización desarrollada en este capítulo.

2.2.3.1 Tipos de requisitos

Aunque se han propuesto muchas taxonomías sobre tipos de requisitos (por ejemplo, [138, 174, 244, 265, 275]), creemos que existe un cierto consenso en que la documentación de los requisitos debe incluir:

- Los *requisitos funcionales*, es decir, los requisitos relativos a las capacidades y servicios que el sistema o el software debe proporcionar.
- Los *requisitos no funcionales*, relativos a los atributos de calidad del sistema y/o del software en el desempeño de sus funciones: por ejemplo, rendimiento, disponibilidad, portabilidad o seguridad.
- Las *restricciones de diseño*, que son los condicionantes existentes para el diseño (pero que no deben ser una anticipación del mismo).

Las reglas de negocio [204] restringen los procesos de negocio describiendo políticas y restricciones sobre la estructura y el comportamiento de los objetos de negocio. Algunos autores incluyen las reglas de negocio dentro de los requisitos funcionales, mientras que otros prefieren considerar que las reglas de negocio no son propiamente requisitos sino más bien restricciones del entorno en el que se va a desplegar el sistema.

2.2.3.2 Identificación y consenso de requisitos

La correcta identificación de interlocutores, la adecuada gestión de los procedimientos para extraer la información necesaria y el establecimiento de los mecanismos que ayuden a desarrollar las discusiones y posteriores acuerdos en la identificación y consenso (*elicitation, extracción*) de los requisitos son factores esenciales para que el proyecto se realice con garantías de éxito.

Las técnicas a emplear en la identificación y consenso de requisitos (ver por ejemplo [82, 265, 296]) incluyen reuniones informales, entrevistas (con o sin cuestionario), mera observación, estudio de documentación existente, talleres tipo FAST (*Facilitated Application Specification Techniques*), o alguna de sus variantes, como JAD (*Joint Application Development*), reuniones tipo "tormenta de ideas" (*brainstorming*), y técnicas de observación más sofisticadas basadas en etnografía.

En combinación con las técnicas anteriores se pueden usar prototipos (ver por ejemplo [245, 250, 296]) para ayudar a extraer requisitos. Para este fin, el prototipo ayuda a conseguir los siguientes beneficios:

- Permite construir rápidamente los diseños de pantallas e informes, incluso interactivamente con los usuarios, permitiendo visualizar claramente los formatos del sistema final.
- Mejora la comunicación entre los usuarios y el equipo de desarrollo de software.
- Acelera la construcción del sistema definitivo, ya que es más fácil construir un software operativo partiendo de un modelo *visible* que de unas especificaciones sobre el papel.
- Absorbe muchos de los nuevos requisitos que se le ocurren al usuario. No sólo ayuda a producir respuestas, sino también preguntas.

- Un documento de requisitos basado en un prototipo tiende a experimentar menos cambios, acortando así el tiempo de desarrollo.

2.2.3.3 Documentación de requisitos

Uno de los objetivos esenciales de la aplicación de un proceso de IR es la consecución de una documentación de requisitos de calidad. En los siguientes apartados se presenta la documentación de los requisitos desde tres puntos de vista:

- La documentación o especificación de los distintos tipos de requisitos.
- La estructura del *Documento de Requisitos del Proyecto* (PRD, *Project Requirements Document*), que puede estar constituido por un conjunto de documentos de requisitos relacionados jerárquicamente.
- La estructura interna o plantilla de cada uno de los documentos de requisitos que componen el PRD.

2.2.3.4 Especificación de requisitos individuales en la documentación de requisitos

Un requisito de calidad debe ser no ambiguo, completo, consistente y verificable [138]. La especificación de cualquiera de los distintos tipos de requisitos se realiza habitualmente en forma textual mediante lenguaje natural, aunque puede complementarse con tablas, figuras o expresiones matemáticas (por ejemplo, fórmulas o expresiones lógicas). En ocasiones, para especificar los requisitos también se usan modelos de casos de uso [60, 153] (incluyendo diagramas de casos de uso, que como señala Larman [174], en realidad juegan un papel complementario dentro del modelo de casos de uso, que es de naturaleza fundamentalmente textual). Los casos de uso se utilizan especialmente en análisis y diseño orientado a objetos, si bien se pueden utilizar también en enfoques tradicionales como la descomposición funcional [201]. Otros modelos muy utilizados para realizar la especificación de un sistema o de una porción de software se construyen mediante ciertas técnicas de UML (*Unified Modeling Language*) [44] utilizadas en el diseño orientado a objetos que se trasladan con éxito a la especificación de los requisitos. Entre las técnicas de diseño –orientado a objetos y convencional– que se utilizan para especificar requisitos se pueden mencionar (1) modelos de datos, como los diagramas entidad-relación [69] o los diagramas de clases de UML; (2) modelos funcionales o de procesos, como los diagramas de flujo de datos [298] o los diagramas de actividades de UML; y (3) modelos dinámicos o de comportamiento, como las máquinas de estados o los diagramas de secuencia de UML. En ocasiones, el lenguaje natural es sustituido completamente en la especificación de los requisitos por alguno o algunos de estos modelos, que tienen asociada una representación gráfica que ayuda a gestionar la complejidad de los requisitos del sistema y/o del software.

Si un objetivo del proyecto de desarrollo de software en cuestión es alcanzar cierto grado de automatismo en la generación de software o bien trabajar con sistemas empotrados o críticos en algún sentido, se hace necesaria una representación formal de los requisitos. Sin embargo, el lenguaje natural se prefiere ampliamente para la especificación y reutilización de requisitos en proyectos de sistemas de información que involucran cuestiones meramente gerenciales o comerciales [82]. El lenguaje natural es

la forma de representación de requisitos más usada por las principales normas en esta materia, como la *Guía para el desarrollo de especificaciones de requisitos de sistema*, IEEE 1233 [139] o la *Práctica recomendada para especificaciones de requisitos de software*, IEEE 830 [138]. Muchos de los interesados (*stakeholders*) de corte no técnico prefieren esta forma de especificar requisitos, dado que facilita la lectura, análisis y tratamiento de los requisitos individuales, y por tanto favorece la participación. Para escribir buenos requisitos y evitar la ambigüedad inherente al lenguaje natural se suele usar una especie de *estilo contractual* para escribir los requisitos (en inglés, a veces se referencian como *system-shall requirements*). Por ejemplo, a este respecto pueden utilizarse las recomendaciones proporcionadas por las dos normas de IEEE mencionadas anteriormente y la referencia de Wiegers [296]. En la Sección 4.1 se comentan los problemas que aparecen con las listas de requisitos textuales cuando el tamaño del sistema es medio o grande.

Además de la mera descripción textual, los requisitos pueden incluir atributos [138, 139], que proporcionan una útil metainformación sobre cada requisito. Por ejemplo, la Norma IEEE 1233 [139], recomienda atributos para los requisitos como identificación (única para cada requisito), prioridad, criticidad, viabilidad, riesgo y fuente. De forma equivalente, otras aproximaciones de IR proponen el uso de una plantilla (por ejemplo Volere [265]).

Un tipo de metainformación de los requisitos particularmente interesante procede de la noción de *trazabilidad*. Para la gestión de requisitos, siguiendo el estándar IEEE 830 [138], la trazabilidad se refiere a la claridad para determinar el origen de cada requisito y a la facilidad de referenciar cada requisito en el desarrollo futuro o en la documentación mejorada. La trazabilidad también se suele definir como la capacidad de describir y seguir la pista de la vida de un requisito en dos direcciones: hacia delante y hacia atrás. Mediante las relaciones de traza se puede comprender toda la vida de un requisito; desde su origen, a través de su desarrollo y especificación, hasta su despliegue y uso, incluyendo su refinamiento. Una referencia clave sobre trazabilidad de requisitos es la de Ramesh y Jarke [256].

2.2.3.5 Estructura jerárquica del documento de requisitos del proyecto

El documento de requisitos del proyecto (PRD, *Project Requirements Document*) constituye la expresión formal de los requisitos del sistema y del software del proyecto. El PRD se divide habitualmente en una colección jerárquica de documentos. En la Figura 2 se muestra una posible organización del PRD, que adaptamos de la propuesta de Gabb [110]. Cada documento dentro de la jerarquía se corresponde con un nivel diferente de especificación y, por tanto, tiene diferentes objetivos y usuarios [110]. No hay un nombre ni estructura estándar para el PRD completo: organizaciones distintas pueden tener diferentes estructuras y diferentes nombres para el mismo, tales como *Documento de requisitos* o *Especificación funcional* (aunque contenga también requisitos no funcionales). A veces parte del PRD recibe un nombre específico, como *Documento visión*, que usualmente incluye las necesidades de usuario y el documento de requisitos del producto.

Para establecer un proceso sistemático de gestión de requisitos la organización de desarrollo debe adoptar una definición del PRD, que debe incluir:

- El tipo y número de documentos de requisitos.
- Los tipos de requisitos que puede contener cada documento del PRD (por ejemplo, requisitos del sistema, requisitos del software, requisitos tipo "caso de uso" –que realmente contienen otros requisitos funcionales y no funcionales–, pruebas de aceptación, etc.). Cada uno de estos tipos de requisitos puede tener su propia metainformación asociada.
- La definición de las relaciones de traza entre los distintos tipos de requisitos usados en el proyecto.

Es preferible que definición del PRD se convierta en una norma en la organización, aunque la configuración de documentos puede variar de un proyecto a otro en función de las necesidades o características específicas de cada uno. En cualquier caso, la estructura de los documentos de requisitos para un proyecto particular debe ser clara desde el principio.

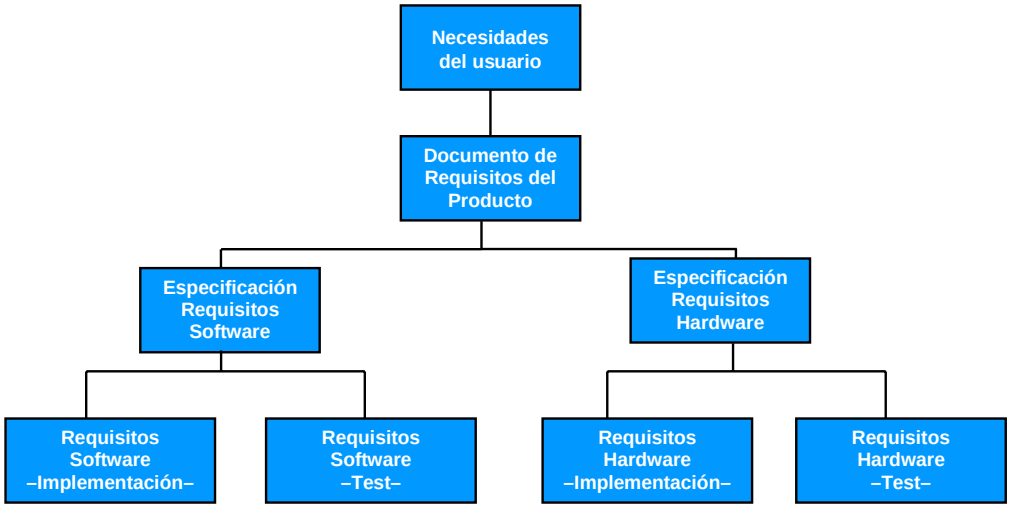

Figura 2 Una posible organización jerárquica del PRD

2.2.3.6 Plantillas de documentos de requisitos

Para la estructura interna de ciertos documentos del PRD existen estándares, a diferencia de lo que ocurre con la estructura completa del PRD. En particular, para el documento de Especificación de Requisitos del Software (ERS, muy conocido por sus siglas en inglés SRS, *Software Requirements Specification*), el estándar más difundido es el IEEE 830 [138], aunque se han propuesto muchos otros a lo largo de los años.

Para la estructura de los documentos de requisitos hay también muchos enfoques no oficiales, algunos bastante extendidos, como las plantillas Volere de Robertson y Robertson [21, 265], que pueden ser usadas de forma gratuita. En el ámbito nacional, otras plantillas de difusión gratuita están disponibles en la Universidad de Sevilla, ligadas a la herramienta de requisitos REM (*REquirements Management*) y su metodología asociada [89, 91].

2.2.3.7 Problemas relacionados con la elaboración del documento de requisitos del proyecto

Con independencia de las normas y/o plantillas utilizadas, en la confección del PRD suelen aparecer problemas de carácter práctico. Destacamos aquí los siguientes:

- Dificultades en el análisis y manejo de gran cantidad de requisitos. Las herramientas CARE y la consideración sólo de los requisitos de interés ayudarán en este punto.
- Consideración de requisitos "obvios". En un documento de requisitos no se trata de especificarlo todo. El sentido común deberá ayudar a establecer las fronteras entre los requisitos obvios y aquellos que no lo son y merecen ser incluidos en el documento.
- Determinación del nivel de detalle en que debe expresarse un requisito. Relacionado con el punto anterior (evitando detallar cuestiones que sean obvias) y con el nivel de abstracción de que se trate (requisitos del sistema –o negocio– frente a requisitos del software). Las relaciones de traza entre requisitos también sirven para ligar descripciones abstractas con descripciones más detalladas.
- Determinación de la ubicación de un requisito en el apartado idóneo del documento de requisitos. A veces un mismo requisito pertenece a varias categorías y puede por tanto ser asignado a una o más secciones del documento. El uso de referencias cruzadas puede ser de utilidad en este punto.

Un problema práctico que creemos merece especial atención se refiere a cómo conjugar el uso de documentos de requisitos con un proceso de desarrollo de software que, como es ampliamente reconocido actualmente, con carácter general debe ser iterativo e incremental. Además, como recogen Kaindl *et al.* [158], la visión de los requisitos como contrato con los clientes ha perdido vigencia para muchos desarrolladores debido al incremento de los desarrollos de productos o paquetes software (*off-the-shelf software*) en detrimento de los desarrollos a medida: en los primeros es el mercado –y no propiamente el cliente– el que dirige los requisitos.

El enfoque de la documentación de requisitos del proyecto presentado en apartados anteriores entra en conflicto con el promovido por los métodos ágiles [2], en los cuales el interés se centra más en el software ejecutable que en su diseño o documentación, que se reducen al mínimo, y la especificación, desarrollo y entrega se realizan de forma iterativa e incremental. En la práctica, los principios en los que se sustentan los métodos ágiles pueden ser difíciles de llevar a cabo. Por ejemplo, como señala Sommerville [275], cuando el cliente confía el desarrollo de software en un proveedor externo, el documento de requisitos del proyecto es usualmente parte del contrato entre el cliente y el proveedor. Dado que la especificación incremental es inherente a los métodos ágiles, escribir un contrato para este tipo de desarrollo puede ser difícil. En un contrato para un desarrollo ágil el cliente paga por el tiempo requerido para el desarrollo del sistema más que por el desarrollo de un conjunto específico, detallado, de requisitos. Si aparecen problemas durante el desarrollo, pueden aparecer disputas para dilucidar quién es el responsable y quién debe hacerse cargo del tiempo y los recursos extra necesarios para resolver los problemas. Para Sommerville [275] los métodos ágiles son adecuados para el desarrollo de productos software y de sistemas de gestión de tamaño pequeño o medio. No son adecuados en el desarrollo de grandes sistemas con equipos de desarrollo

distribuidos en distintas localizaciones, ni donde existen interacciones complejas con otros sistemas software y hardware, ni en el desarrollo de sistemas críticos en los cuales es necesario realizar un análisis detallado de de todos los requisitos del sistema para comprender todas las implicaciones de seguridad.

Es ampliamente reconocido que los requisitos no son inamovibles y no se puede pretender "congelarlos". Sin embargo, siguiendo el enfoque presentado en este apartado, basado en un documento de requisitos, en algún momento hay que aprobar el documento de requisitos, aún asumiendo que éste pueda sufrir cambios en el futuro. El momento de esta aprobación por parte del cliente se llama *punto de acuerdo*, *firma* o *sign-off*. El conjunto de requisitos aprobados y acordados en un punto en el tiempo (normalmente en el punto de acuerdo) se denomina *requisitos base*. Wiegers [296] propone un texto para la firma final: "Reconozco que este documento representa nuestra mejor comprensión de los requisitos para este proyecto al día de hoy y constituye un punto de arranque para el resto del proyecto. Se podrán realizar cambios a este punto de inicio siguiendo el proceso de cambios definido para el proyecto. Igualmente reconozco que los cambios aprobados en un futuro podrían requerir una renegociación de costes, recursos y compromisos de planificación temporal para este proyecto".

2.2.4 Enfoques alternativos de ingeniería de requisitos

Sin ánimo de ser exhaustivos, en este apartado se identifican algunas estrategias no convencionales para IR, y se comentan brevemente los aspectos más significativos de cada una de ellas.

2.2.4.1 El enfoque orientado a objetivos

En relación con la IR orientada a objetivos (*GORE, Goal-Oriented Requirements Engineering*) destacan los trabajos de van Lamsweerde *et al.* sobre KAOS y los de Yu *et al.* sobre i*. En el enfoque del grupo de van Lamsweerde [290] se distinguen los "objetivos" del sistema ("*goals*") de los requisitos propiamente dichos. Los objetivos son aquellas metas que el sistema debería alcanzar, y pueden ser formulados a diferentes niveles de abstracción (estratégicos o técnicos). Los objetivos pueden ser funcionales y no funcionales y normalmente se necesitan varios agentes para su realización. En este contexto un requisito se define como una meta que para ser realizada sólo necesita un agente o tipo de agente dentro del sistema (cuando el único agente no es del sistema, sino de su entorno, entonces la meta se denominará *suposición* o *hipótesis*). Los agentes se definen en este contexto como los componentes activos del sistema, como por ejemplo las personas, el software o determinados instrumentos. Una vez construido un modelo de objetivos, éstos pueden ser relacionados con modelos concretos de objetos, como entidades, relaciones y/o agentes. Los objetivos pueden representarse de manera informal, semiformal (usando texto y/o gráficos) y formal (mediante una lógica temporal lineal para tiempo real), lo que permite la verificación formal y la gestión de conflictos de las especificaciones. La representación formal es opcional y está reservada para los aspectos más críticos del sistema. KAOS es un método orientado a objetivos que proporciona un lenguaje de especificación de requisitos y los elementos necesarios para describir los objetivos, requisitos, suposiciones y demás conceptos dentro de este enfoque, incluyendo soporte automatizado (*Objectiver*).

El enfoque i*, propuesto por Yu [12, 299], constituye básicamente un enfoque orientado a agentes para IR. El enfoque está centrado en la intencionalidad del agente (el nombre i* se refiere a la idea de "intencionalidad distribuida"). Los agentes se atribuyen propiedades intencionales (tales como objetivos, opiniones, capacidades y obligaciones) y razonan sobre relaciones estratégicas. Las dependencias entre agentes dan lugar tanto a oportunidades como a vulnerabilidades. Las redes de dependencias son analizadas utilizando técnicas de razonamiento cualitativo. Los agentes consideran configuraciones de dependencias alternativas para evaluar su posicionamiento estratégico en un contexto social determinado. Este enfoque puede ser utilizado especialmente en contextos en los existen múltiples partes (o unidades autónomas) con intereses estratégicos que pueden reforzarse o entrar en conflicto entre sí. Ejemplos de tales contextos son la reingeniería de procesos de negocio, el rediseño de empresas, la IR para sistemas de información, el análisis de la adopción social de las TIC o el diseño de sistemas software basados en agentes.

2.2.4.2 El enfoque basado en puntos de vista

La construcción de un sistema complejo implica a muchos interesados. Cada uno de ellos tiene una visión parcial o incompleta del sistema, ligada al papel (o responsabilidades) que tiene en el mismo. Según Finkelstein [102], se define *punto de vista* (*viewpoint*) como la combinación de un *agente* con la *vista* que tiene del sistema. El enfoque basado en puntos de vista ayuda a organizar la información atendiendo a las distintas perspectivas y fuentes del sistema y también ayuda a poner de manifiesto rápidamente la existencia de conflictos entre los agentes. Esto implica la realización de un proceso de resolución de conflictos que cada uno de los métodos de este grupo resuelve de distinto modo. La información completa sobre los requisitos es derivada mediante la integración de los puntos de vista. Los métodos basados en puntos de vista consideran a éstos como elementos de primera clase en su concepción del sistema.

Kotonya y Sommerville [165] analizan métodos de gestión de requisitos basados en puntos de vista como SADT (*Structured Analysis and Design Technique*), CORE (*COntrolled Requirements Expression*), VOSE (*Viewpoint-Oriented System Engineering*) y VORD (*Viewpoint-Oriented Requirements Definition*).

2.2.4.3 El enfoque basado en aspectos

Este enfoque, que en cierta forma se podría relacionar con la estrategia de puntos de vista, se basa en extrapolar a los requisitos ideas y conceptos tratados en programación orientada a aspectos (AOP, *Aspect Oriented Programming*). Un *interés* o *incumbencia* (*concern*) es todo aquello que resulta importante para una aplicación, una cuestión a la que es preciso prestar atención para resolver un problema (por ejemplo, las funciones del sistema, la persistencia, la seguridad, etc.). Cuando se elige una determinada descomposición para un problema, hay un conjunto de incumbencias que aparecen diseminadas en el código atravesando partes del sistema no relacionadas en el modelo: son las *incumbencias transversales* (*cross-cutting concerns*). Un *aspecto* es básicamente una incumbencia transversal modularizada, un módulo software que no puede ser encapsulado en un procedimiento. Un aspecto no se corresponde con una unidad funcional del sistema, sino con una propiedad que afecta la ejecución o la semántica de los componentes.

En el proceso de IR es preciso especificar las relaciones entre requisitos y razonar sobre ellas, para ser capaces de detectar posibles inconsistencias y de arbitrar soluciones a los conflictos que se presentan entre las necesidades contradictorias de los interesados. Por tanto, es necesario disponer de técnicas que permitan separar las distintas incumbencias de los interesados, en el nivel de los requisitos, con el fin de poder razonar por separado sobre ellas e integrarlas después. Es habitual que en el proceso de IR se escoja una manera de realizar la partición de los requisitos en la cual ciertas incumbencias no se puedan ubicar adecuadamente. Por ejemplo, los requisitos no funcionales no tienen una correspondencia exacta con una descomposición funcional, basada en casos de uso o en puntos de vista. Si se realiza una descomposición funcional, ciertas cuestiones de seguridad pueden afectar transversalmente varias funciones del sistema. Las reglas de negocio exhiben un comportamiento parecido. Todos estos requisitos se pueden considerar por tanto incumbencias transversales. La cuestión reside entonces en la especificación individual de estos intereses que cruzan transversalmente la jerarquía de documentos de requisitos, especificando las relaciones con otros intereses y permitiendo su composición, de manera que se pueda razonar sobre posibles interacciones e identificar posibles conflictos. Con el enfoque orientado a aspectos se define una estrategia de obtención de requisitos basada en *aspectos de requisitos* (*early aspects*) que facilita la traducción de los mismos a un entorno de programación AOP y que puede ser aplicada también a otros entornos más generales. Rashid [257] proporciona una reciente introducción en este campo de la IR.

2.2.4.4 Modelado de requisitos no funcionales

Los requisitos no funcionales suelen estar muy relacionados con preocupaciones inmediatas de los usuarios (facilidad de uso, disponibilidad, seguridad, rendimiento, etc.). Existen propuestas que abordan de forma explícita el modelado de los requisitos no funcionales. Entre ellas destaca la propuesta NFR de Chung *et al.* [72], en la que se plantea el uso de los requisitos no funcionales como guía en el proceso de desarrollo del software. Con este enfoque se hacen explícitos los objetivos funcionales y no funcionales del sistema y las relaciones entre ellos, que pueden ser de distintas clases (sinérgica, colaborativa, contradictoria, etc.).

2.2.5 Herramientas de gestión de requisitos

Las herramientas de gestión de requisitos son herramientas CASE (*Computer-Aided Software Engineering*) que facilitan el trabajo de los ingenieros de requisitos permitiendo la especificación, organización, almacenamiento y gestión de los requisitos. A través de la gestión de trazas, estas herramientas también ayudan a evaluar el posible impacto de los cambios en los requisitos en los procedimientos, costes y personal. En esta memoria de tesis nos referiremos a estas herramientas con el acrónimo *CARE* (*Computer-Aided Requirements Engineering*), si bien ciertos autores prefieren denominarlas *bases de datos de gestión de requisitos* (*RMDB tools, Requirements Management Data Base tools*) o simplemente *herramientas de gestión de requisitos* (*RM tools, Requirements Management tools*).

En resumen, las herramientas CARE deben proporcionar operaciones de creación, edición y recuperación de requisitos y de gestión de trazas, además de funcionalidades como las siguientes:

- Informes de alto nivel, incluyendo cuestiones como número de requisitos, análisis de impacto –coste estimado de los cambios propuestos–, requisitos que pueden ser afectados por un cambio, estado de desarrollo del proyecto (por ejemplo, número de requisitos implementados y/o probados), matrices de traza, etc.
- Plantillas para la documentación de los requisitos del proyecto.
- Integración con otras herramientas CASE.
- Posibilidad de construir extensiones personalizadas (*add-ins* o *plugins*).

En la práctica, para gestionar los requisitos los desarrolladores confían en muchas ocasiones en herramientas ofimáticas tradicionales como MS Word, MS Excel o MS Access. Matulevicius [211] afirma que tal situación es debida a que el mercado de CARE es muy amplio y en muchas ocasiones no está claro qué herramienta CARE es mejor adquirir, con lo cual es difícil justificar la carga financiera de su adquisición. Este autor plantea un marco de evaluación de CARE denominado R-TEA (*RE-Tool Evaluation Approach*), junto con el prototipo de una herramienta que permite la aplicación de R-TEA para evaluar la adecuación de las herramientas CARE candidatas a los requisitos que impone el proceso de IR que se pretende automatizar en algún sentido.

El mercado de herramientas CARE es muy extenso. INCOSE (*International Council on Systems Engineering*) mantiene un *survey* de más de cuarenta herramientas CARE [144]. A pesar de que se trata de un listado bastante completo y siendo probablemente el más referenciado, se le podrían añadir herramientas CARE como Borland Caliber DefineIT [7] (que junto con la conocida Borland Caliber-RM [6] constituye la suite Borland Caliber Analyst), y herramientas de *arquitectura del negocio* (*EA, Enterprise Architecture*) que incorporan gestión de requisitos, tales como Enterprise Architect de Sparx Systems [10], ARIS Solutions de IDS Scheer [5], Holocentric Modeler [14], Metastorm ProVision [16] y Casewise Modeller [8]. Otros listados reseñables de herramientas CARE son los de Volere [292], Alexander [22] y Ludwig Consulting Services [17].

Entre las herramientas CARE, las dos que destacaríamos por ser tradicionalmente bien conocidas y de uso común en la comunidad internacional de IR son IBM Rational Requisite Pro [18, 260] e IBM Telelogic DOORS (*Dynamic Object-Oriented Requirements System*) [9]. Se trata de dos herramientas presentes en todas las comparativas entre herramientas que conocemos. Tras la compra de Telelogic en 2008, será interesante observar la estrategia que sigue IBM en relación con estas dos herramientas. De momento ambas se mantienen como productos diferenciados. Requisite Pro es una herramienta orientada a documentos, de manera que los requisitos se almacenan en una base de datos pero ligados en tiempo real con el archivo MS Word en el que reside el documento de especificación de requisitos (si bien los requisitos se pueden almacenar también directamente en la base de datos de la herramienta). De esta forma se ofrece al analista un entorno sencillo y familiar –MS Word– para definir y organizar los requisitos. En cambio DOORS está orientada a un repositorio de requisitos y no a documentos. En DOORS los requisitos se consideran objetos y los documentos se tratan como módulos. Estas dos herramientas se proporcionan con un interfaz de extensibilidad que permite añadir nuevas funcionalidades, y tienen desarrollada una versión web además de las versiones multiusuario de escritorio.

En el ámbito nacional merece ser destacada la herramienta IRqA (*Integral Requisite Analyzer*) [145], de Visure Solutions, un *spin-off* de TCP Sistemas e Ingeniería fundado en 2007, que cuenta con presencia en el mercado nacional e internacional, incluyendo la definición de pruebas de aceptación, soporte gráfico a la modularidad de requisitos y recientemente ciertas características de reutilización. The Reuse Company [20] es otra organización española que proporciona extensiones para las herramientas DOORS e IRqA con el fin de soportar reutilización, desde una perspectiva de gestión de conocimiento y de ontologías. También en el ámbito nacional, pero en un contexto más académico, se puede citar la herramienta REM [89] de la Universidad de Sevilla.

2.3 Reutilización y requisitos

El propósito de la reutilización de requisitos es identificar descripciones de sistemas que puedan ser reutilizadas (en su totalidad o en parte) con un mínimo número de modificaciones, de manera que se reduzca el esfuerzo total de desarrollo [67]. Como hemos justificado en la Sección 1.1.2, creemos que es éste el nivel de reutilización que aporta mayores beneficios. En esta tesis doctoral nos centramos en la reutilización de requisitos, si bien la reutilización *temprana* o en el nivel de análisis (*early reuse*) está abierta a la reutilización de cualquier *artefacto de análisis* (*early artifact*), como planes de gestión del proyecto, planes de gestión de la calidad, pruebas de validación, etc.

Como afirman Robertson y Robertson [265], "cualquier diagrama (de contexto, de clases, de casos de uso) o especificación puede ser utilizado para hacer los requisitos visibles. De manera que cualquiera de estas fuentes puede ser reutilizable". En este Capítulo 2 estudiamos enfoques de reutilización de requisitos que consideran de forma más o menos explícita la representación de requisitos en lenguaje natural, que es la forma más utilizada en la industria para especificar requisitos y la técnica de especificación de requisitos más vinculada con los contenidos de este capítulo. Así pues en esta Sección 2.3 nos centramos en la reutilización de requisitos escritos en lenguaje natural y no pretendemos, por tanto, establecer una taxonomía completa de reutilización de requisitos. Enfoques que quedan fuera de este apartado son, por ejemplo, los trabajos clásicos de Maiden y Sutcliffe (como [195]) sobre reutilización por analogía (que analizan modelos del dominio expresados en una notación de objetos, con el fin de encontrar el modelo más adecuado para un dominio dado), y otros enfoques basados en analogía que utilizan lenguajes formales en lugar de lenguaje natural. Es importante remarcar que dejamos fuera de esta sección (1) las propuestas de reutilización que se centran en los requisitos no funcionales, en particular en cuestiones como seguridad o protección de datos personales, que forman parte de una tesis doctoral en curso dentro del GIS, la de Joaquín Lasheras; y (2) aquellos aspectos de las propuestas de reutilización de requisitos que muestran explícitamente los metamodelos de requisitos subyacentes, pues forman parte de otra tesis doctoral en curso dentro del GIS, la de Begoña Moros.

Los trabajos revisados han sido seleccionados entre los resultados de ejecutar las cadenas de búsqueda "reusing requirements", "reuse of requirements" y "requirements reuse" en los siguientes motores de búsqueda:

- IEEE Digital Library (www.computer.org/portal/site/csdl/index.jsp)

- ACM Digital Library (portal.acm.org)
- Science@Direct (www.sciencedirect.com)
- MetaPress (Kluwer+Springer) (www.metapress.com)
- Wiley InterScience (www.interscience.wiley.com)
- CiteSeer (citeseer.ist.psu.edu)
- Google Scholar (scholar.google.com)

En esta sección analizamos un conjunto de trabajos relacionados con la reutilización de requisitos en lenguaje natural, que se sintetizan en la Tabla 11. Para comprender los campos *Gen.* y *Com.* de la Tabla 11 se debe tener en cuenta que en la literatura se distinguen dos vías no disjuntas a la reutilización: enfoques *basados en generación* y enfoques *basados en composición* (ver, por ejemplo, von Knethen *et al.* [293]): (1) los enfoques basados en generación se centran en la instanciación de abstracciones reutilizables, como por ejemplo en líneas de productos o en la instanciación de patrones; y (2) los enfoques basados en composición se centran en componer componentes reutilizables: se establecen relaciones de traza entre los requisitos entre sí y entre los requisitos y la arquitectura, implementación y pruebas que se construyen a partir de ellos, de manera que si un requisito del sistema es reutilizado, se puede acelerar el proceso de desarrollo posterior siguiendo las trazas y componiendo los elementos trazados. En reutilización de requisitos tenemos ejemplos de ambos enfoques, si bien quizás los enfoques generativos son más frecuentes [293].

Esta Sección 2.3 se divide en apartados para facilitar su lectura, pero es preciso remarcar que las secciones no son disjuntas, dado que algunos trabajos se pueden clasificar desde distintos puntos de vista. Otras revisiones interesantes de la literatura de reutilización de requisitos son por ejemplo (1) la de Bühne *et al.* [47], que incluye una revisión de la reutilización de requisitos centrada en líneas de productos; y (2) clasificaciones de enfoques generales de reutilización de requisitos, anteriores a la de Bühne, como la de Lam *et al.* [169] y las de Cybuslky [63, 65, 67].

2.3.1 Reutilización de requisitos en lenguaje natural

Cybulsky estudia desde distintas perspectivas la reutilización de requisitos en lenguaje natural. A pesar de que se trata de trabajos relativamente antiguos ya, los incluimos en esta revisión del estado del arte porque creemos que son trabajos importantes en la reutilización de requisitos en lenguaje natural. Cybulsky [63] propone un proceso de reutilización de requisitos basado en las siguientes etapas: (1) análisis de requisitos informales (identificación, representación, generalización); (2) organización en un repositorio de artefactos reutilizables (clasificación, almacenamiento, recuperación); y (3) síntesis de nuevos documentos de requisitos de componentes reutilizables (selección, adaptación, composición, integración). Cybulsky *et al.* [66] identifican casi un centenar de artefactos de análisis reutilizables, que son analizados y comparados de acuerdo con siete dimensiones: medio electrónico utilizado, estructura, función, objeto/sujeto, productor/consumidor y valor para la reutilización. El objetivo es calcular el *grado de afinidad* entre pares de artefactos de análisis, que se usa para agrupar los artefactos en grupos de artefactos similares que en principio pueden compartir los métodos útiles en su reutilización.

Enfoque	Año	Método	Gen.	Com.	Modelo de procesos	Técnicas y guías	Organización de requisitos	Herramienta	Validación
Cybulsky *et al.* [63-67]	1996-2000	RARE (*Reuse-Assisted RE*)	■	■	Sí, básicamente análisis, organización y síntesis	Patrones de procs de reut de artefactos de análisis; taxonomía de artefactos de análisis; trazas con diseño	Catálogos de dominio, por palabras clave, facetas, tesauro	IDIOM	(CEA) Experimentos académicos
Lam *et al.* [169-172]	1996-1998	FORE (*Family Of REquirements*)	■	■	Sí, en aspectos técnicos (FORE): establecer grupo, análisis del prod, estructuración de reqs, generación de sists, análisis de cambio; en aspectos gerenciales: marco *ORM: Operational Reuse Model* junto con KPF, *Key Practices Framework*	Reqs *plantilla* (parametriz), con procedimientos para especificar parámetros de reqs; patrones de reqs (que incluyen reqs típicos relacionados en un dominio); trazas de dependencia entre reqs	Familias de prod; incluye árbol de variabilidad de req y discriminantes; org por docs de req	COMPASS	(CEI) Control motores aviones en Rolls-Royce University Tech. Centre para Rolls Smiths Engine Controls
Snook *et al.* [274]	2008	Event-B	■	□	Sí, para obtener un *mod genérico* (parametrizado): análisis del dominio e ingeniería del dominio	Validación y verificación de *reqs genéricos* (parametriz) usando E/R, UML-B y B	LDP	U2B, ProB, y RM *-Reqs Manager*	(CEI) Sists de control de motores de aviones en ATEC Engine Controls
Mannion *et al.* [197, 199]	1999-2008	MRAM (*Method for Requirements Authoring and Management*)	■	■	Sí, para modelar la línea de prods y para instanciar prods: construir mod de la familia de aplicaciones; construir mod del sistema	MM de requisitos explícito con diccionario del dominio; parámetros y *discriminantes* (exclusivo, alternativo y opcional); trazas (padre/hijo)	LDP; reqs org según ptos de vista de los interesados	TRAM	(CEI) Control de naves espaciales y (CEA) teléfono móvil
Faulk [98]	2001	PRS, *Product-Line Requirements Specification*	■	□	Sí, para obtener un PRS bien definido	Familias de programas (*Synthesis*), métodos formales para reqs en sists empotrados (*SCR*) y métodos de especificación OO (*CoRE*); mods de decisión; téc de análisis variabilidad	LDP; PRS	—	(CEI) LDP de sist de control de vuelo comercial Rockwell Collins
Dueñas *et al.* [56, 217]	2004-2005	—	■	■	Se proporcionan guías pero no un modelo de procs explícito, se centra en el MM de requisitos	Guías prácts para la reut de reqs en DOORS; MM explicito de los reqs (incluye reqs parametriz)	LDP; FRD (*Functional Requirements Documents*)	DOORS [217] y ENAGER [56]	(CEI) LDP en sists de control en aviones
Djebbi Salinesi [85]	2007	RED-PL (*Requirements Elicitation & Derivation for Product Lines*)	■	□	Sí, para la derivación de reqs del prod: extracción reqs cli; correspondencia con reqs LDP, derivación conjunto óptimo reqs prod	Características (aplicable a cualquier técnica); dependencias entre reqs (*composition, requires, optional composition, exclusion, alternative*); técnicas de análisis de similitud	LDP	Ms Excel	(CEI) Analizadores de sangre en Stago Instruments
Tavakoli Reiser [279]	2007	—	■	□	—	Características; MM de variabilidad de reqs; reqs abstractos y técnicos; reqs parametriz ; trazas entre reqs (*is_independent*, *includes*, *excludes*, *influences*)	LDP; biblioteca de reqs	Extensión a DOORS con dxl	(CEI) Unidades electrónicas de control en Daimler Chrysler
von Knethen *et al.* [293]	2002	—	■	■	Sí, seis pasos para el reciclado sistemáticos	Mod conceptual del sist y mod conceptual de documentación	Basada en docs	—	(CEI) Sists empotrados en Daimler Chrysler
Alexander Kiedaisch [23]	2002	—	□	■	Sí, para reciclar reqs: arqueología, preparación y reciclado	Cdu; dependencias de reqs vert. y horiz.	Base de reqs (base de datos o doc)	DOORS / Scenario Plus	(CEI) Door Control Unit Daimler Chrysler
Robertson Robertson [264, 265]	1996-2006	Volere	□	■	Sí, para selecc patrones de req: establecer contexto, comparar patrones con contexto, construir especificación	Mod de procs y de datos para cada patrón de reqs, definido por partición de eventos	Libro de patrones con patrones de reqs	—	(CEA) Biblioteca
Laguna *et al.* [168]	2004	—	■	■	Sí, para la transformación entre requisitos	MM de requisitos explicito que soporta requisitos que implican interacción, como casos de uso y flujos de trabajo	Repositorio GIRO, organizado por dominios mediante *mecanos*	R^2	(CEI) Herramientas para discapacitados

Leyenda: Enfoque: *Gen*. Basado en generación; *Com*. Basado en composición; LDP: Línea De Productos; MM: Metamodelo; OO: Orientado a Objetos; CEI: Caso de Estudio Industrial / CEA: Caso de Estudio Académico

Tabla 11 Síntesis de enfoques de reutilización de requisitos relacionadas con el lenguaje natural

En un estudio previo del estado del arte en reutilización de requisitos, Cybulski [65] examina la manera en la que se crean y usan los documentos de requisitos con el fin de construir un "marco de reutilización de requisitos" en forma de un conjunto de "patrones para reutilización de requisitos". Cada patrón describe informalmente un método para reutilizar un artefacto de análisis involucrado. Por ejemplo, estos patrones incluyen estándares y plantillas de documentos, y reutilización de modelos del negocio, de frases de texto, de comentarios y anotaciones, de patrones de especificación y de trazas de validación. Estos patrones describen métodos de reutilización de requisitos basados en las propiedades estructurales de los documentos de requisitos y en los procesos utilizados en su construcción, pero sin embargo no proporcionan técnicas específicas para llevarlos a cabo. Algunos de los patrones propuestos han sido probados por medio de RARE IDIOM (donde *RARE* es el nombre del método, *Reuse-Assisted Requirements Engineering*, e *IDIOM* es el nombre de la herramienta, *Informal Document Interpreter, Organiser and Manager*).

En relación con las herramientas CARE de soporte a la reutilización, Cybulsky [64] estudia la idoneidad de un procesador de textos estándar, como Ms Word, como soporte de la reutilización de requisitos textuales, y concluye que este tipo de aplicaciones tiene las capacidades mínimas necesarias para soportar dicha reutilización, y que, por tanto, la reutilización de requisitos en lenguaje natural puede ser efectiva incluso sin entornos de reutilización especializados. Según el proceso de reutilización de requisitos anteriormente mencionado, Cybulsky clasifica en tres grupos las capacidades requeridas a las herramientas: análisis, organización y síntesis, y muestra a través de un sencillo experimento que las capacidades de un procesador de textos como Ms Word son similares a las de entornos de programación para reutilizar código (como *GNU Emacs*) o especificaciones algebraicas (como *Larch LSL y LP*). En nuestra opinión (desarrollada en las secciones 2.8 y 2.9), un enfoque basado en procesador de textos como herramienta de soporte a la reutilización puede ser suficiente si se trata de llevar a cabo simplemente una reutilización *pura*, reutilizando tal cual porciones de texto, o *referencial*, copiando y pegando una porción de texto y adaptándola al contexto de la reutilización. En cambio, creemos que un enfoque basado en procesador de textos no tiene funciones capaces de gestionar adecuadamente las técnicas implicadas en un proceso de reutilización de requisitos avanzado: gestionar la metainformación de los requisitos (incluyendo las relaciones de traza), la organización de los requisitos (en base de datos, por documentos, por vistas, por aspectos, etc.), las búsquedas avanzadas (que involucran la metainformación), el copiar y pegar "inteligente" de los requisitos a reutilizar (teniendo en cuenta la reutilización de atributos y la reutilización de trazas), la definición e instanciación de requisitos parametrizados, etc. De hecho Cybulsky propone una herramienta para la reutilización de requisitos, la anteriormente mencionada RARE IDIOM, pues termina reconociendo que los procesadores de texto generales adolecen de ciertas carencias como soporte a la reutilización de requisitos. En IDIOM Cybulsky y Reed [67] implementan un mecanismo de clasificación de requisitos y diseños basado en palabras clave, facetas y tesauros, que usan para determinar el grado de similitud entre requisitos.

2.3.2 Reutilización de requisitos en lenguaje natural en líneas de productos

Un grupo importante de propuestas de reutilización de requisitos se enfocan al análisis del dominio, en muchas ocasiones en el marco de líneas de productos. Por ejemplo, en

los trabajos de Lam se puede observar una evolución desde la reutilización de requisitos en lenguaje natural a la reutilización de requisitos en lenguaje natural en el ámbito de líneas de productos. Como los de Cybulsky, son trabajos antiguos pero que nos parecen interesantes: en este caso Lam parte de la idea desarrollada en este Capítulo 2, la reutilización de requisitos en lenguaje natural, que aborda también en general y desde distintas perspectivas, y avanza hacia la idea desarrollada en el Capítulo 3 de esta memoria de tesis doctoral, la reutilización de requisitos en el ámbito de líneas de productos. En los trabajos de Lam se considera explícitamente la representación de requisitos en lenguaje natural, si bien en análisis del dominio son muy utilizadas técnicas diagramáticas, como modelos de características o modelos convencionales de ingeniería del software, con y sin extensiones (en el Capítulo 3 se incluye una visión general de estas técnicas).

Lam *et al.* [172] comienzan argumentando que la reutilización sistemática se debe estructurar a tres niveles: (1) *dominio* (conocimiento acerca de las funciones y restricciones en un área de aplicación); (2) *componente* (colección de productos reutilizables de desarrollo del sistema); y (3) *artefacto* (productos del desarrollo del sistema en su nivel de granularidad más bajo). Con base en esta estructura, estos autores organizan los requisitos en tres niveles: (1) *dominio*, por medio de una red semántica; (2) *componente*, agrupando *requisitos plantilla* (que nosotros denominamos *parametrizados*) de acuerdo con la función a la que están relacionados, dando lugar a *componentes de requisitos*; (3) *artefacto*, por medio de requisitos parametrizados.

Con base en la experiencia de los autores en el dominio de los sistemas de control de motores de aviones, Lam *et al.* [171] exponen diez pasos prácticos hacia la reutilización de requisitos. Se trata de un trabajo que nos parece especialmente interesante y que como vamos a ver en esta misma sección ha sido referenciado por otros autores. Estas diez cuestiones clave son las siguientes: (1) “cuidado con las generalizaciones seductoras”, lo cual podemos traducir con otras palabras como “ser selectivo con las generalizaciones” (hay que tener en cuenta que la mayor parte de los requisitos son requisitos de bajo nivel, de grano fino, a menudo ligados con el diseño, y por tanto difíciles de reutilizar); (2) identificar *familias de sistemas* para maximizar la reutilización (los requisitos vienen determinados por el contexto y son específicos a un problema o conjunto de problemas); (3) evaluar la tecnología de reutilización en términos del *cambio de procesos*, no sólo en términos del potencial de reutilización (la tecnología que ofrece el máximo potencial no es siempre la más adecuada para la organización, por ejemplo puede ser demasiado costosa o tener asociados demasiados riesgos); (4) las *cuestiones del dominio* (*domain issues*) actúan como puntos de enfoque o áreas de interés de los requisitos, y pueden ser usadas para organizar y estructurar los productos y los procesos de la reutilización (los autores introducen la noción de *cuestión –issue–* como un mecanismo de estructuración, un área donde los requisitos se suelen enfocar dentro de un dominio determinado); (5) una *abstracción justificada* (*reasoned abstraction*) es efectiva para desarrollar requisitos parametrizados (los autores describen un método para crear requisitos de este tipo); (6) después de trabajar en un dominio particular a menudo emergen *patrones de requisitos* (que engloban distintos requisitos relacionados por trazas); (7) hacer explícito el contexto de la reutilización para prevenir su mal uso; (8) partes del proceso de reutilización son también reutilizables; (9) factorizar las variaciones de los requisitos en *partes de requisitos conectables* (*pluggable requirements parts*); y (10) valorar de antemano el impacto de la

reutilización de requisitos en la cadena de producción, pues cambios en el proceso de IR pueden afectar el diseño y la prueba.

En la línea marcada por la cuestión clave (2) anterior, Lam [169] se centra en el estudio de la reutilización de requisitos en familias de productos, describiendo el método *FORE* (*Family Of REquirements*), cuya principal contribución es una guía metodológica de la reutilización en el nivel de análisis (*early reuse*). FORE incluye un modelo de procesos que implica la construcción de un *árbol de variabilidad* y la generación de un documento de requisitos específico dentro de la familia de productos. Los requisitos se organizan por medio de dicho *árbol de variabilidad*, que es similar a un árbol de características (Sección 3.3.2.1) pero en el nivel de los requisitos. Este árbol de variabilidad debe reflejar la estructura de las trazas existentes entre requisitos. Lam sugiere que un documento FORE debe contener (como mínimo): (1) visión general del producto; (2) estudio de factibilidad; (3) árbol de variabilidad; (4) requisitos parametrizados principales (*template-core requirements*, requisitos "estándar" o "típicos" en un área funcional del sistema, que están parametrizados para capturar la variabilidad); (5) historial de uso; e (6) historial de cambios. Un aspecto a destacar es que FORE no requiere para su implantación sustituir completamente el proceso de IR que pueda existir en la organización.

Finalmente, Lam *et al.* [170] estudian distintos enfoques a la reutilización de requisitos, y encuentran escasa evidencia de su aplicación en la industria, concluyendo que pocas organizaciones saben cómo organizar su proceso de reutilización. Estos autores identifican factores en la transferencia de tecnología de reutilización, y subrayan el hecho de que la mayor parte de la literatura de reutilización se concentra en cuestiones técnicas, por lo que deciden centrarse en las cuestiones de gestión. Una idea que proponen y que creemos acertada es que la reutilización se debe considerar como un aspecto más de un programa de mejora de la madurez del proceso de desarrollo. La principal contribución de este trabajo es la presentación de un *Modelo de reutilización operacional* (*Operational Reuse Model, ORM*) y de un *Marco de prácticas clave* que soportan los procesos de ORM (*Key Practices Framework*, KPF). Estas prácticas se refieren a aspectos tales como el análisis de oportunidad de la reutilización, coste-beneficio, métricas, etc., aspectos que según estos autores están poco presentes en la literatura sobre reutilización. Tanto ORM como KPF no son presentados, creemos que acertadamente, como una solución definitiva, cerrada, sino como unos procesos e ideas que las organizaciones de desarrollo pueden adaptar y mejorar para su propio uso.

El trabajo de Snook *et al.* [274] se puede considerar una extensión de los trabajos de Lam anteriores. Estos autores estudian formalmente la instanciación de requisitos *genéricos* (parametrizados) en el ámbito de líneas de productos. La organización de los requisitos del dominio sigue el consejo de Lam, y se hace en términos del conocimiento del dominio, utilizando un *modelo genérico*. El acento se pone en verificar los requisitos parametrizados y el proceso de instanciación. Las instancias de cada requisito genérico se expresan como datos en forma tabular, a partir de los cuales se construye un primer modelo Entidad/Relación, que se traslada a un modelo UML-B, que a su vez se traslada a una especificación formal en B que se puede animar.

Mannion *et al.* han trabajado en la especificación de líneas de productos software mediante requisitos en lenguaje natural [197, 199]. Estos autores proponen MRAM (*Method for Requirements Authoring and Management*), un método que permite la

definición de una línea de productos y su instanciación en un producto determinado. Un modelo del dominio viene dado por requisitos en lenguaje natural, un diccionario del dominio y un conjunto de *discriminantes* que permiten diferenciar un sistema de otro. Los requisitos se relacionan entre sí mediante relaciones padre/hijo o a través de discriminantes, que pueden ser de tres tipos: (1) exclusión mutua; (2) lista de alternativas; y (3) opción. MRAM dispone de una herramienta, *TRAM*, que permite recorrer los discriminantes e ir adoptando las decisiones oportunas en la instanciación.

En un trabajo interesante como el de Bühne *et al.* [47] se referencia a Faulk [98], que en el ámbito de los sistemas críticos embebidos investigan un campo similar al de von Knethen *et al.* [293], la derivación del *documento de requisitos del producto* (*Product-line Requirements Specification*, *PRS*) en líneas de productos a través de una estrategia sistemática de análisis de partes comunes y variables. El proceso propuesto incluye: (1) organizar la definición del dominio (una jerarquía de requisitos en lenguaje natural); (2) crear un modelo de decisión (una tabla que especifica las decisiones implícitas en el modelo del dominio); (3) encapsular variaciones en *CoRE* (un tipo de modelo de clases que representan los requisitos en lenguaje natural); (4) definir variaciones; y (5) proporcionar variabilidad. Sin embargo, los autores no proporcionan un proceso para derivar una Especificación de Requisitos del Software (ERS) a partir del PRS.

Monzón y Dueñas [217] señalan la ausencia de soluciones prácticas a la reutilización de requisitos en herramientas CARE comerciales y presentan guías para reutilizar requisitos en DOORS. Los autores analizan los *Documentos de requisitos funcionales* (*Functional Requirements Documents*, FRD) y definen un metamodelo que clasifica los requisitos contenidos en tales documentos. Más tarde, Cerón *et al.* [56] trabajan en requisitos de familias de sistemas y modelizan los aspectos involucrados en la reutilización de requisitos en líneas de productos (trazabilidad, separación de requisitos comunes y específicos, representación explícita de la variabilidad, selección de variabilidad, gestión de decisiones y conflictos, reutilización y evolución de requisitos). Estos autores presentan una herramienta, *ENAGER*, *Environment for Requirements Analysis and Management*, que soporta la definición de los conceptos del metamodelo (glosarios, participantes, sistemas, requisitos, casos de uso, conflictos, decisiones, clases de prueba, artefactos, etc.).

Djebbi y Salinesi [85] también presentan un trabajo sobre derivación de requisitos del producto en líneas de producto software, pero desde un punto de vista distinto al de von Kneuthen *et al.* [293] y Faulk [98], que se centraban en documentos. RED-PL (*Requirements Elicitation & Derivation for Product Lines*) es un método cuyo objetivo es soportar la derivación de requisitos, contrastando los requisitos del producto indicados por los usuarios con los requisitos de la línea de productos (a través de modelos de características), y derivando una especificación de requisitos. Este trabajo se puede clasificar desde su aportación en la reutilización de requisitos en el ámbito de líneas de productos o desde su relación con trabajos de generación de modelos de requisitos textuales a partir de modelos del dominio –este trabajo aparece en la revisión sistemática de la literatura sobre generación de especificaciones de requisitos textuales a partir de modelos de ingeniería del software presentada en la Sección 4.2, en la cual se proporciona una visión más completa de los trabajos de Djebbi–.

Tavakoli y Reiser [279] se proponen mejorar el enfoque de líneas de producto gestionando la variabilidad en el nivel de los requisitos a través de una *biblioteca de*

requisitos. Para ello estos autores extienden la expresividad de los modelos de características y definen técnicas para generar especificaciones de requisitos desde la biblioteca de requisitos. La biblioteca de requisitos consiste fundamentalmente de requisitos abstractos y de requisitos técnicos que han sido abstraídos a partir de decisiones de diseño. La reutilización de requisitos dentro de la biblioteca de requisitos es dirigida por la selección de características, que preselecciona requisitos de la base de datos. Los autores presentan una extensión de DOORS que soporta los modelos de variabilidad. (Al igual que la anterior de Djebbi y Salinesi, esta contribución se analiza también en la Sección 4.2 en el ámbito de la revisión sistemática citada en el párrafo anterior.)

2.3.3 Reciclado de requisitos en lenguaje natural

von Knethen *et al.* [293] trabajan en el análisis del dominio en sistemas de automoción y complementan los trabajos de Lam de instanciación de requisitos parametrizados a través de lo que denominan *reciclado de requisitos* en un enfoque basado en documentos de requisitos. Estos autores trabajan en el *reciclado* de requisitos, la actividad de extraer requisitos de documentos de requisitos existentes e insertarlos en un nuevo documento de requisitos que describe un producto similar (es habitual que una organización especifique cada producto en un documento de requisitos separado). El reciclado de requisitos ocurre frecuentemente en la industria, pero normalmente es realizado de forma *ad hoc*. Sin embargo, según estos autores los enfoques no sistemáticos (1) son propensos a error y consumen mucho tiempo; (2) es difícil identificar todos los requisitos que pueden ser reutilizados, pues los documentos de requisitos tienen una estructura variable; (3) a menudo no se copian todos los requisitos relacionados con un requisito copiado, porque las dependencias entre requisitos no son explícitas; y (4) a veces ocurre lo contrario, y se copian demasiados requisitos. Por todo esto en el nuevo documento de requisitos del producto se introducen omisiones, inconsistencias y características superfluas. Estos autores presentan una vía mixta, basada en generación y en composición. Una plantilla, que incluye las entidades de los documentos existentes que van a ser reciclados, proporciona la faceta basada en generación, mientras que el aspecto basado en composición viene dado por las guías para documentar explícitamente las relaciones de los requisitos reutilizables y cómo usar tales relaciones para copiar correctamente candidatos al reciclado. Los autores comentan que soportan reutilización como hace Lam *et al.* [171], a través de plantillas, pero Lam *et al.* se dirigen a la reutilización de grano fino, palabras en sentencias, mientras que este trabajo se dirige a la reutilización de grano grueso, similitudes entre documentos en el mismo dominio. En el ámbito de la automoción, este enfoque define las relaciones entre las entidades "lógicas" del sistema (a través de un *Modelo conceptual del sistema*) y las entidades "de documentación" del sistema (a través de un metamodelo de documentos, *Modelo de documentación conceptual*).

Alexander y Kiedaisch [23] también hablan de "reciclado" de requisitos, que entienden como la selección de requisitos adecuados de una *base de requisitos* y su integración en una nueva especificación de requisitos. En esta propuesta los requisitos en lenguaje natural se agrupan en casos de uso y se relacionan por medio de trazas verticales y horizontales. Las trazas verticales representan relaciones de refinamiento entre casos de uso y especificaciones de requisitos del sistema (a distintos niveles de abstracción). Las trazas horizontales representan relaciones entre elementos en el mismo nivel de

abstracción. La propuesta no dispone de soporte automatizado pero sí de modelo de procesos.

2.3.4 Patrones de requisitos

El éxito de los patrones de diseño de Gamma *et al.* [111] ha llevado a trasladar este concepto a otros ámbitos del desarrollo de software, como la especificación. Por ejemplo, son conocidos los *patrones de análisis* de Fowler [108], que son básicamente modelos de clases a nivel conceptual que describen determinados dominios recurrentes en el desarrollo de software.

Lam *et al.* [171] (Sección 2.3.2) hablan de patrones de requisitos en su cuestión clave (6). Estos autores se refieren a patrones de requisitos en un sentido distinto al de requisitos parametrizados, requisitos plantilla o requisitos genéricos: con patrones de requisitos hacen referencia a estructuras de requisitos relacionados que se presentan recurrentemente en un dominio. El nivel de abstracción, como vemos, es más alto que el nivel del texto.

Robertson y Robertson [264, 265] ligan el concepto de reutilización con el de *patrón de requisitos*. Estos autores definen los conceptos de *patrón de proceso de requisitos* y de *patrón de datos de requisitos*, que abstraen respectivamente el proceso y los datos relacionados con un caso de uso o evento del sistema (siguiendo la idea clásica de McMenamin y Palmer de *partición de eventos* en el análisis estructurado). Estos patrones se guardan en un *Libro de patrones* que es consultado (por nombre del patrón) cuando se inicia un nuevo análisis y se debe analizar un nuevo evento. Siguiendo esta "partición de eventos" se pueden agrupar los requisitos (de proceso y de datos) en el contexto relacionado con un evento/caso de uso. Como vemos este enfoque no es exclusivo de requisitos en lenguaje natural, pero lo hemos seleccionado en esta revisión del estado del arte pues se enmarca en un método como Volere que está íntimamente relacionado con los requisitos en lenguaje natural.

2.3.5 Transformaciones de requisitos

Como vamos a ver esta propuesta ya no se restringe a requisitos en lenguaje natural. Laguna *et al.* [168] muestran un enfoque de reutilización basado en estandarización y transformaciones de requisitos con el objetivo de facilitar las búsquedas de requisitos especificados con distintos formatos. En el Grupo GIRO de la Universidad de Valladolid se trabaja en reutilización basada en unidades reutilizables complejas de grano grueso denominadas *mecanos*, que están formadas por unidades de grano más fino denominadas *assets*, definidos en cada mecano a tres niveles de abstracción: requisitos, diseño e implementación. Los requisitos constituyen la puerta de acceso a los mecanos almacenados en el repositorio. En una primera fase de la investigación, en el nivel de los requisitos se utilizaba lenguaje natural que incorporaba los patrones lingüísticos propuestos por Durán *et al.* [90]. Además de los problemas inherentes al lenguaje natural, Laguna *et al.* constatan que "la diversidad de formatos de requisitos es una restricción para su reutilización." Por tanto, en una segunda fase deciden generalizar la representación de los requisitos a varias técnicas de especificación, teniendo en cuenta especialmente los requisitos que implican interacción con el usuario (como casos de uso, casos de uso del negocio, flujos de trabajo –*workflows*– o diagramas de actividad UML). Todas estas representaciones comparten un metamodelo común que

facilita la transformación de unas representaciones a otras (por ejemplo, de casos de uso a flujos de trabajo), con el fin de facilitar la comparación de assets en el repositorio. La representación interna de los requisitos se realiza mediante una notación de flujos de trabajo formalizada a su vez por redes de Petri.

2.4 El método SIREN

SIREN [226, 280, 281, 283, 284] es un enfoque práctico basado en reutilización y en estándares de ingeniería del software que ayuda a seleccionar y especificar los requisitos de un sistema software. En principio, los requisitos de SIREN se escriben en lenguaje natural, pero pueden incluir todo tipo de objetos como información complementaria del propio requisito –como por ejemplo tablas, diagramas o esquemas de cualquier tipo–.

El método SIREN incorpora un modelo de procesos en espiral, guías de aplicación de dicho proceso, un repositorio reutilizable de requisitos y una herramienta CARE (*Computer-Aided Requirements Engineering*) de soporte, denominada *SirenTool* [177, 226, 280]. El repositorio de requisitos reutilizables de SIREN se encuentra organizado por *catálogos*, que se corresponden con un conjunto de requisitos reutilizables. En SIREN estos catálogos se pueden corresponder con (1) *dominios verticales* o simplemente *dominios*, como contabilidad o finanzas; o (2) *dominios horizontales* o *perfiles*, conjuntos de requisitos aplicables de forma transversal a distintos dominios (como protección de datos de carácter personal [281, 285] o seguridad [281, 284]). Estos catálogos, a su vez, están organizados en una jerarquía de documentos de especificación de requisitos que siguen una estructura de acuerdo a estándares del IEEE. La creación de los catálogos formaría parte del proceso *para* reutilización (*SIRENp*), mientras que el uso de tales catálogos formaría parte del ciclo *con* reutilización (*SIRENc*, Figura 3).

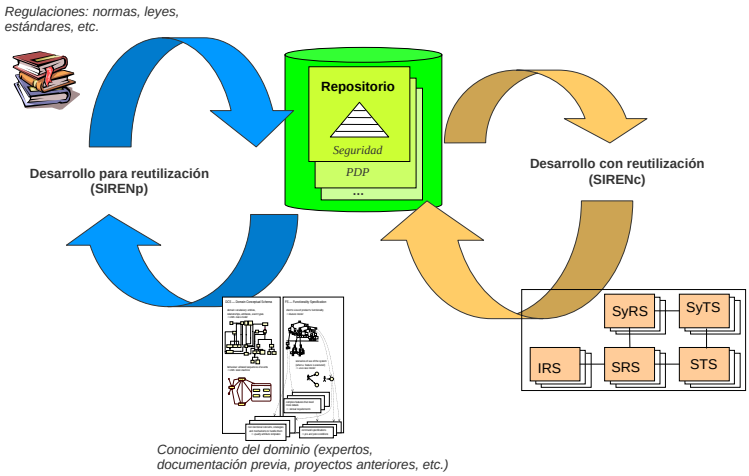

Figura 3 Ciclos con y para reutilización

Wiegers [295] afirma que más que nuevos métodos, lo que hace falta para acercar el estado de la práctica y el estado de la investigación en IR es hacer los métodos existentes más atractivos y menos engorrosos: en definitiva, facilitar su puesta en práctica. Aunque existen muchas técnicas para abordar la reutilización de requisitos (véase por ejemplo la Sección 2.3 o la clasificación de Cybulsky y Reed [67]), SIREN

se centra en la reutilización de requisitos desde un punto de vista muy práctico, intentando que su uso sea lo más sencillo posible.

Más que para ser aplicado propiamente como un método completo de IR, SIREN se ha concebido para ayudar a la reutilización de requisitos en lenguaje natural en el marco de cualquier método de desarrollo de software. A modo de ejemplo, en la Sección 2.4.1 se muestra la compatibilidad de SIREN con la Metodología de planificación y sistemas de la administración pública española, Métrica 3. No obstante, SIREN también puede ser utilizado como un primer método de IR en una organización de desarrollo en la que no se haya definido un método de IR, y para ello incorpora un conjunto básico de guías generales de IR.

En las siguientes secciones se describe en profundidad el método SIREN, siguiendo a Nicolás *et al.* [226], documento en el que se recoge la evolución de SIREN (versión 2.0) a partir de la experiencia en el proyecto GARTIC (descrito en la Sección 1.4.3.1). Los elementos que conforman la descripción de SIREN son los siguientes: (1) en la Sección 2.5 se estudia el modelo de procesos de SIREN; (2) en la Sección 2.6 se estudian las guías que SIREN proporciona para la definición de requisitos, para la reutilización de los mismos, para la organización de los requisitos y para la mejora del catálogo; (3) en la Sección 2.7 se muestra la experiencia acumulada en la aplicación de SIREN; y (4) en la Sección 2.9 muestra brevemente la herramienta de soporte. Después, en la Sección 2.10 se plantea SIRENgsd, una extensión a SIREN para adaptarlo al desarrollo global de software. Finalmente, la Sección 2.11 recopila las conclusiones acerca de esta línea de investigación de reutilización de requisitos.

2.4.1 SIREN y Métrica 3

Como se ha comentado anteriormente, SIREN se ha concebido más como una especie de extensión o *add-in* sobre el método de IR definido en la organización de desarrollo que desea considerar la reutilización en el nivel de los requisitos, que como un método de IR completo, si bien en la descripción de SIREN se proporcionan un modelo de procesos, técnicas y guías como para que pueda ser usado como primer método de IR en aquellas organizaciones con poca madurez en su gestión de los requisitos.

Como ejemplo, podemos afirmar que SIREN es compatible con Métrica 3 [201], pues SIREN proporciona soporte completo a la actividad ASI 2, "Establecimiento de Requisitos" del proceso de Análisis del Sistema de Información (ASI) de Métrica 3, a través de una propuesta concreta de estructuración del catálogo de requisitos de Métrica 3 y de un modelo de procesos para llevar a cabo las tareas de ASI 2 de (1) "Obtención de requisitos" (ASI 2.1); (2) "Análisis de requisitos" (ASI 2.3); y (3) "Validación de requisitos" (ASI 2.4). SIREN en cambio no proporciona guías específicas para ASI 2.2, "Especificación de casos de uso", que es una tarea obligatoria en el itinerario de análisis orientado a objetos de Métrica 3 y opcional en el itinerario estructurado.

Además, a través de los catálogos de requisitos presentados en la Sección 2.7.1, SIREN también proporciona apoyo en las actividades de Métrica 3 que introducen la seguridad en todo el ciclo de vida. En concreto, en la tarea DSI 1.7. "Especificación de Requisitos de Operación y Seguridad" del proceso Diseño del Sistema de Información (DSI), así como en el Interfaz de Seguridad de Métrica 3 con MAGERIT 1.0 (especialmente en las

actividades ASI-SEG 3, “Especificación de Requisitos de Seguridad” y DSI-SEG 2, “Especificación de Requisitos de Seguridad del Entorno Tecnológico Previsto”).

2.5 Modelo de procesos de SIREN

Un *proceso de software* o *proceso del software* (*software process*) es un conjunto de actividades cuya meta es el desarrollo o evolución del software [275]. Un *modelo de procesos del software* (*software process model*) o simplemente *modelo de procesos* es una descripción simplificada de un proceso del software, que como cualquier labor de modelado, se realiza desde una perspectiva particular [275]. Lo que se modela en un modelo de procesos, es decir, el proceso del software, es un conjunto de actividades y artefactos asociados que producen un software. En esta sección se comienza revisando un modelo de procesos de IR básico (Sección 2.5.1), para extenderlo después con las actividades de reutilización de requisitos propias de SIREN (Sección 2.5.2).

2.5.1 Un modelo de procesos básico de IR

Como se resumía en la Sección 1.1.1, siguiendo a Kotonya y Sommerville [165] las actividades típicas de un proceso general de IR son las siguientes:

1. *Identificación y consenso de requisitos (elicitation).* En esta actividad los requisitos se “descubren” haciendo uso de técnicas de recogida de información (Sección 2.2.3). Otras denominaciones para esta actividad son por ejemplo extracción, adquisición o descubrimiento de requisitos.
2. *Análisis y negociación de requisitos.* En esta actividad los requisitos se analizan en detalle. Los interesados negocian para llegar a un acuerdo sobre qué requisitos van a ser incluidos en el sistema. Esta actividad es necesaria, pues es inevitable que aparezcan conflictos entre requisitos que proceden de distintas fuentes y porque en ocasiones la información es incompleta o los requisitos identificados son incompatibles con el presupuesto disponible para el desarrollo.
3. *Documentación (especificación) de requisitos*. Los requisitos sobre los que se ha alcanzado un acuerdo en la actividad anterior se documentan, con el nivel de detalle apropiado para que sean comprensibles por parte de todos los interesados, al tiempo que son útiles al equipo de desarrollo. Generalmente los requisitos se documentan usando lenguaje natural y técnicas diagramáticas (ver Sección 2.2.3).
4. *Validación de requisitos.* Los requisitos documentados se deben comprobar cuidadosamente para garantizar la consistencia de los mismos, su completitud y precisión. El objetivo de esta actividad es detectar problemas en el documento de requisitos antes de que sea utilizado como base para el desarrollo del sistema. Todos los actores que han colaborado en las actividades anteriores han de jugar un papel importante en la validación de los requisitos.

Kotonya y Sommerville [165] afirman que un ciclo de vida en espiral (ver Figura 4) es la manera más adecuada para abordar el proceso de IR, dado que no es posible obtener

un conjunto completo de requisitos en una sola iteración. Las actividades de IR se repiten hasta que se adopta la decisión de que el documento de requisitos sea aceptado. Si se encuentra algún problema en el borrador del documento de requisitos se vuelve a comenzar la espiral con las mismas actividades de IR. Este proceso continúa hasta que se produce un documento de requisitos aceptable o hasta que determinados factores externos tales como presiones en los plazos de entrega o falta de recursos obligan a dar por bueno el documento de requisitos en su estado actual.

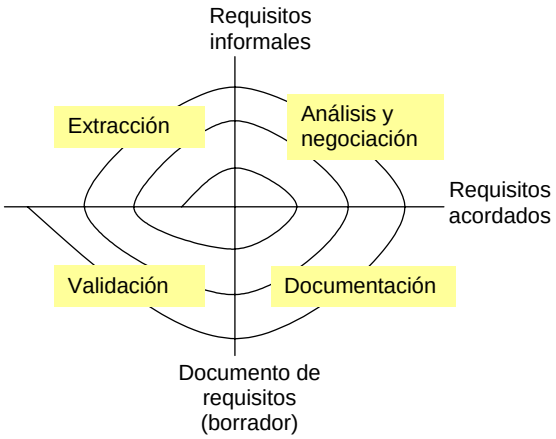

Figura 4 Modelo en espiral de un proceso general de IR

2.5.2 El modelo de procesos de SIREN

En SIREN se asume el modelo de procesos general en espiral de Kotonya y Sommerville [165] mostrado en la Figura 4, que se enriquece con nuevas actividades para abordar específicamente las cuestiones relacionadas con la reutilización de requisitos. Como se ha señalado en la Sección 2.1, SIREN se concibe como un método de requisitos que puede aportar a cualquier método de IR la consideración explícita de la reutilización. En la Figura 5 se muestra cómo las actividades de SIREN relacionadas con la reutilización (señaladas en rojo) extienden el modelo de procesos de la Figura 4.

La Figura 6 especializa la Figura 5 para mostrar la especificación del modelo de procesos de SIREN utilizando la notación estándar SPEM (*Software Process Engineering Metamodel*) [237]. En dicho modelo aparecen las actividades del modelo de procesos de SIREN (en adelante denominadas *tareas* siguiendo la terminología de SPEM) junto con los roles (agentes) responsables de llevarlas a cabo (el modelo de roles de SIREN se muestra en la Sección 2.5.3). El modelo de procesos *para* reutilización se representa por la tarea T1, mientras que el modelo de procesos *con* reutilización se representa en la espiral que conforman las restantes tareas.

Para adaptar SIREN al método de IR que se haya implantado en la organización de desarrollo, las tareas específicas de SIREN (que son T1, T2, T3 y T8 en la Figura 6) se deberían ubicar en el modelo de procesos del método de IR en cuestión. Para cada una de las tareas de la Figura 6 se describe en la Sección 2.5.4 un conjunto de buenas prácticas que son de utilidad para su correcta ejecución. El orden de estos listados de buenas prácticas no determina necesariamente el orden en el que deben llevarse a cabo, puesto que en la práctica el trabajo se realiza de forma iterativa, con continuos solapamientos y retroalimentaciones.

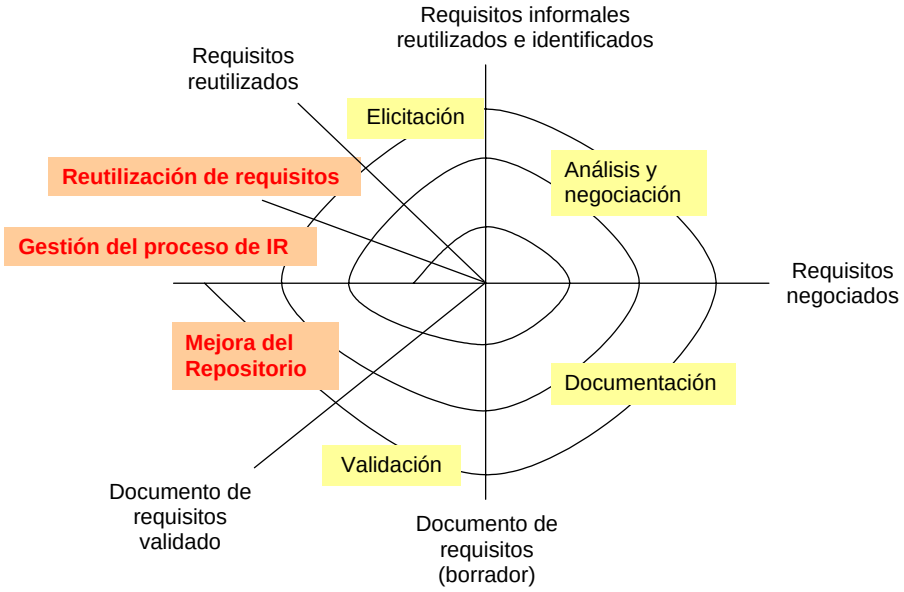

Figura 5 Relación de las actividades de SIREN con un modelo de procesos general de IR

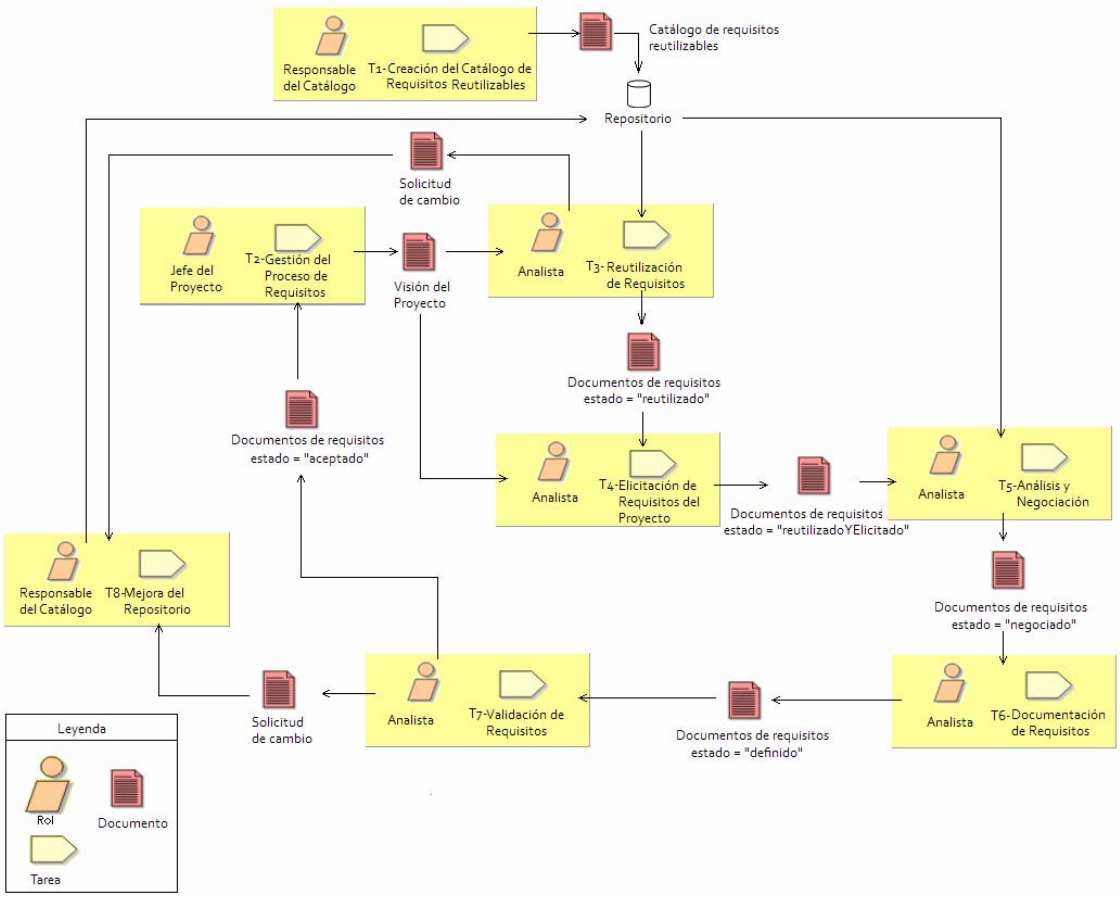

Figura 6 Modelo de procesos de SIREN con la notación SPEM

En la Sección 2.6 se recogen las pautas de SIREN para abordar las tareas propias de la reutilización. En la descripción de las tareas se detallan las entregas resultado de las

mismas. La estructura de los documentos de requisitos que aparecen en la Figura 6 se detalla en la Sección 2.6.8.

Para mayor información acerca de las tareas generales de un proceso de IR se puede consultar Wiegers [296], a quien seguimos en buena medida en la definición de las cuestiones generales de SIREN, en especial de las buenas prácticas recogidas en las tareas del modelo de procesos (Sección 2.5.4).

2.5.3 Modelo de roles de SIREN

La Tabla 12 muestra los roles que se han definido en SIREN, que representan los actores o agentes que desempeñan un cierto papel en el modelo de procesos del método. Evidentemente una misma persona puede jugar distintos roles y un rol puede ser jugado por distintas personas durante el proceso de requisitos. Dado que SIREN es un método que normalmente extenderá el modelo de procesos de IR propio de la organización de desarrollo, los roles de SIREN serán un subconjunto de los roles implicados en dicho modelo de procesos general.

Rol	Descripción
Responsable del catálogo	Encargado de la coordinación de las actualizaciones de un catálogo. Para no sobrecargar de trabajo a una sola persona, en ocasiones este rol es jugado por distintas personas. El grupo de personas que desempeñan este rol constituye el *Equipo responsable del catálogo*.
Jefe de proyecto	Encargado de coordinar el desarrollo del proyecto con reutilización.
Analista o Ingeniero de Requisitos	Desempeña labores propias de analista o ingeniero de requisitos en cualquier proyecto de desarrollo de software.
Usuario clave	Conoce muy bien todo el dominio o una parte del mismo y aporta el conocimiento necesario para elaborar los documentos de requisitos. Es seleccionado de una clase de usuarios del cliente *equivalentes* para ayudar en la extracción y validación de requisitos, debido a su mayor conocimiento del dominio, su mayor jerarquía en la organización cliente o su mayor disponibilidad.
Usuario típico	Conoce el dominio o una parte del mismo porque ha sido usuario de versiones anteriores del producto o de productos similares y por tanto puede aportar conocimiento útil para elaborar los nuevos documentos de requisitos.

Tabla 12 Modelo de roles de SIREN

2.5.4 Tareas de un modelo de procesos general de ingeniería de requisitos con SIREN

En este apartado se desarrollan las tareas del modelo de procesos de IR definido con SIREN que es reflejado en la Figura 6.

2.5.4.1 T1-Creación del catálogo de requisitos reutilizables

Objetivos: En el marco de un desarrollo *para* reutilización, se trata de desarrollar un catálogo de requisitos reutilizables para un dominio o perfil determinado.

Roles implicados:

- Responsable:
  - Responsable del catálogo

- Participantes:
  - Analista
  - Usuarios clave

Buenas prácticas:

- Definir el ámbito de alcance del catálogo.
- Establecer la jerarquía de documentos de requisitos apropiada para el dominio o perfil (Sección 2.6.5).
- Adoptar una plantilla para cada documento de requisitos seleccionado (Sección 2.6.8).
- Determinar los *niveles de requisitos* (Sección 2.6.1) más adecuados para el dominio o perfil.
- Identificar las características (*features*) de las aplicaciones del dominio o perfil.
- Crear una partición del dominio o perfil en *módulos de análisis*: escoger una descomposición por funciones, modos, objetos de negocio, estímulos (partición de eventos), etc. (Sección 2.6.8)
- Definir requisitos para cada una de las características.
- Asignar requisitos a módulos de análisis.
- Identificar *clusters* de requisitos (Sección 2.6.4.3).
- Identificación de *puntos de variación* (Sección 2.6.6).
- Especificación de *requisitos parametrizados* (Sección 2.6.3).
- Identificar y establecer las relaciones de traza entre requisitos, y entre requisitos y posibles documentos (ver atributo *fuente* en la Sección 2.6.2 y las relaciones de traza en la Sección 2.6.4).
- Generalizar requisitos potencialmente reutilizables.

Entregas:

- Catálogo de requisitos reutilizables, que se añade al repositorio.

2.5.4.2 T2-Gestión del proceso de requisitos

Objetivos: En el marco de un desarrollo *con* reutilización, se trata de planificar y monitorizar la ejecución del proceso de IR, controlar las desviaciones que se puedan producir durante dicha ejecución y si procede generar lecciones aprendidas que puedan ser de utilidad en futuros desarrollos.

Roles implicados:

- Responsable:
  - Jefe de proyecto

- Participantes:
  - Analista
  - Usuarios clave

Buenas prácticas:

- Establecer el modelo de procesos de IR (en el marco del cual se ubica SIREN) que debe dirigir el proyecto.
- Definir el ámbito de alcance del proyecto en el que se debe desarrollar el sistema o producto.
- Determinar los niveles de requisitos (Sección 2.6.1) más adecuados para el proyecto.
- Determinar los requisitos del negocio, es decir, las necesidades estratégicas del negocio, para determinar una *visión del proyecto* (ver Entregas en esta misma tarea).
- Identificar las características (*features*) del proyecto.
- Crear una partición del proyecto en módulos de análisis, por ejemplo basada en funciones, modos, objetos de negocio, estímulos (partición de eventos), etc.
- Identificar clases de usuarios y sus habilidades o destrezas.
- Seleccionar un *usuario clave* para cada clase de *usuario* (ver Tabla 12).
- Establecer grupos de *usuarios típicos* (ver Tabla 12) de versiones anteriores del producto o sistema, a partir de los cuales se pueden extraer nuevos requisitos.
- Derivar iterativamente planes y calendarios de trabajo (no sólo de IR sino del resto de desarrollo del software) a partir de los requisitos.
- Renegociar las obligaciones del proyecto (planes y calendarios) cuando los requisitos cambian.
- Documentar y gestionar los riesgos relacionados con los requisitos.
- Registrar el esfuerzo dedicado a la IR.
- Revisar las lecciones aprendidas en otros proyectos relacionadas con los requisitos.
- Generar las nuevas lecciones aprendidas en este proyecto, si las hubiera.

Entregas:

- Modelo de procesos de IR para utilizar en el proyecto.
- *Visión del proyecto*, que incluye ámbito de alcance del proyecto, requisitos de negocio y/o de sistema y descomposición en módulos de análisis.
- Registro de las clases de usuarios y de los usuarios clave para cada una de ellas.
- Registro de grupos de usuarios típicos.
- Planes, calendarios de trabajo y riesgos identificados.
- Registro de recursos (tiempo, personal, etc.) utilizados en el proceso de IR.

- Lecciones aprendidas (en su caso).

2.5.4.3 T3-Reutilización de requisitos

Objetivos: Se trata de descubrir y seleccionar los requisitos reutilizables del repositorio que son de aplicación en el proyecto actual.

Roles implicados:

- Responsable:
  - Analista
- Participantes:
  - Usuarios clave y típicos

Buenas prácticas:

- Refinar las particiones del proyecto en módulos de análisis.
- Identificar los catálogos reutilizables involucrados en el proyecto.
- Trabajar con los usuarios clave y típicos para la reutilización de características (*features*) y requisitos.
- Definir búsquedas de requisitos.
- Buscar y seleccionar requisitos.
- Determinar la resolución de los puntos de variación (Sección 2.6.6).
- Adaptar al proyecto actual los requisitos reutilizados.
- Identificar requisitos reutilizables susceptibles de ser mejorados.

Entregas:

- Documentos de requisitos en estado “reutilizado”.
- Solicitudes de cambios en el catálogo reutilizable utilizado.

2.5.4.4 T4-Extracción de requisitos del proyecto actual

Objetivos: Se trata de descubrir los requisitos que deben formar parte del proyecto actual y que no se encuentran en los catálogos de requisitos reutilizables disponibles en el repositorio.

Roles implicados:

- Responsable:
  - Analista
- Participantes:
  - Usuarios clave y típicos

Buenas prácticas:

- Escribir la visión del producto o sistema a desarrollar en el proyecto, que se debe incluir en el documento de requisitos de sistema (si existe) o en el apartado correspondiente del documento de especificación de requisitos del software. (A partir de la entrega *Visión del proyecto* de la tarea T2.)
- Refinar las particiones del proyecto en módulos de análisis.
- Trabajar con los usuarios clave y típicos para la identificación de nuevos requisitos no reutilizados.
- Identificar requisitos susceptibles de ser incluidos en el repositorio.
- Identificar eventos y respuestas del sistema.
- Realizar talleres de extracción de requisitos (Sección 2.2.3.2).
- Observar a los usuarios durante la realización de sus tareas.
- Examinar informes de problemas de sistemas actuales para obtener ideas sobre nuevos requisitos.

Entregas:

- Documentos de requisitos en estado “reutilizadoYElicitado”.
- Documento con requisitos susceptibles de ser incluidos en el repositorio.

2.5.4.5 T5-Análisis y negociación de requisitos

Objetivos: Se trata de analizar los conflictos, incompletitudes y ambigüedades que puedan estar presentes en el conjunto de nuevos requisitos reutilizados y extraídos, así como su viabilidad. Este análisis se debe realizar teniendo en cuenta los requisitos que ya han sido validados en los documentos de requisitos en ciclos anteriores del desarrollo iterativo. Por último, se debe proceder a una negociación para resolver los conflictos detectados.

Roles implicados:

- Responsable:
  - Analista
- Participantes:
  - Usuarios clave

Buenas prácticas:

- Dibujar el diagrama de contexto.
- Desarrollar prototipos –funcionales y de interfaz de usuario–.
- Analizar la viabilidad de los requisitos.
- Priorizar los requisitos.
- Modelar los requisitos.

- Crear un diccionario de datos.
- Asignar requisitos a los módulos de análisis.
- Negociar los requisitos en tres etapas: (1) *etapa de negociación*, donde se determina la naturaleza de los problemas asociados a un requisito; (2) *etapa de discusión*, donde analistas y usuarios clave discuten cómo se podrían resolver tales problemas, si es necesario examinando los requisitos presentes en el repositorio en busca de soluciones de compromiso; y (3) *etapa de resolución*, donde se decide qué requisitos pueden ser eliminados, qué requisitos deben ser incluidos en la especificación del sistema, y se sugieren modificaciones específicas para los requisitos, se reasignan prioridades y se cambia el atributo *motivación* (Sección 2.6.2) de los requisitos modificados para reflejar dichos cambios.

Entregas:

- Documentos de requisitos en estado “negociado”.

2.5.4.6 T6-Documentación de requisitos

Objetivos: Se trata de documentar mediante lenguaje natural y/o técnicas diagramáticas los requisitos que han sido acordados durante la negociación anterior.

Roles implicados:

- Responsable:
  - Analista
- Participante:
  - Jefe de Proyecto

Buenas prácticas:

- Establecer la jerarquía de documentos de requisitos apropiada para el proyecto (Sección 2.6.5).
- Adoptar una plantilla para cada documento de requisitos seleccionado (Sección 2.6.8).
- Determinar los tipos de requisitos a utilizar en la especificación (Sección 2.6.1) y revisar los atributos asociados a dichos tipos (Sección 2.6.2).
- Etiquetar de manera única cada requisito.
- Establecer el atributo *fuente* de los requisitos (Sección 2.6.2).
- Documentar requisitos del negocio y del sistema.
- Documentar las características (*features)* del producto o sistema.
- Documentar los requisitos funcionales conforme a los módulos establecidos.
- Documentar las reglas de negocio.
- Documentar los requisitos de calidad del sistema o producto (facilidad de uso, rendimiento, eficiencia, fiabilidad, etc.).

Entregas:

- Documentos de requisitos en estado “definido”.

2.5.4.7 T7-Validación de requisitos

Objetivos: Se trata de revisar nuevamente los documentos de requisitos, en muchos casos junto con los usuarios clave, con el objetivo de eliminar cualquier error que persista en los documentos de requisitos y darlos por válidos.

Roles implicados:

- Responsable:
  - Analista
- Participantes:
  - Usuarios clave

Buenas prácticas:

- Inspeccionar los documentos de requisitos.
- Derivar casos de prueba a partir de los requisitos.
- Definir criterios de aceptación.
- Validar requisitos susceptibles de ser incluidos en el repositorio y elaborar las solicitudes de cambio en el repositorio.

Entregas:

- Documentos de requisitos en estado “aceptado”.
- Solicitudes de cambios en el repositorio.

2.5.4.8 T8-Tarea de mejora del repositorio

Objetivos: Se trata de estudiar posibles mejoras del repositorio de requisitos reutilizables y determinar su inclusión o no en el mismo. Las solicitudes de cambio se generan (1) directamente en la tarea T3, cuando se trata de mejoras de los requisitos ya incluidos en el repositorio, que van directamente al responsable del catálogo; o bien (2) en la tarea T4, debido a que los usuarios clave conocen bien un dominio o perfil del repositorio y descubren nuevos requisitos que no están presentes en el dominio o perfil. Estas últimas solicitudes que se generan en la tarea T4 han de ser validadas en la tarea T7.

Roles implicados:

- Responsable:
  - Equipo responsable del catálogo
- Participantes:
  - Analista

Buenas prácticas:

- Definir un proceso de control de cambios en los requisitos, a través del cual se proponen, analizan y resuelven cambios en los requisitos.
- Realizar el análisis de impacto de los cambios en los requisitos.
- Crear matrices de traza.
- Establecer líneas base y control de versiones de los documentos de requisitos.
- Mantener un histórico de los cambios de los requisitos.
- Medir la *volatilidad de los requisitos*, es decir, el número de cambios propuestos y aprobados sobre los requisitos por unidad de tiempo.

2.6 Guías de SIREN

En esta sección se incluyen guías que ayudan en la ejecución de las tareas del proceso SIREN. Estas guías son las siguientes: (1) definición de requisitos (Sección 2.6.1); (2) definición de atributos (Sección 2.6.2); (3) definición de requisitos parametrizados (Sección 2.6.3); (3) definición de trazas entre requisitos (Sección 2.6.4); (4) organización del catálogo (Sección 2.6.5); (5) reutilización de requisitos (Sección 2.6.6); (6) mejora del repositorio (Sección 2.6.7); y (7) la organización de la documentación de requisitos (Sección 2.6.8).

2.6.1 Guías para la definición de requisitos

Como se decía en la Sección 1.1.1 de esta memoria, según el IEEE 610 (*Standard Glossary of Software Engineering Terminology*) [137], un requisito se define como: "(1) Una condición o capacidad necesitada por un usuario para resolver un problema o alcanzar un objetivo; (2) una condición o capacidad que debe ser satisfecha o poseída por un sistema o componente de sistema para satisfacer un contrato estándar, una especificación u otro documento impuesto formalmente; y (3) una representación documentada de una condición o capacidad como en (1) o en (2)."

Todos los requisitos no se refieren siempre a un mismo nivel de abstracción en la descripción del espacio del problema. Existen distintos *niveles de requisitos*, entre los cuales destacan los siguientes:

- *Requisitos del negocio* (*business requirements*, necesidades estratégicas o de alto nivel del negocio): representan objetivos de alto nivel del cliente o de la organización que solicita el sistema.
- *Requisitos del sistema (system requirements)*: describen los requisitos de alto nivel de un sistema. Un sistema puede estar compuesto completamente de software o puede incluir subsistemas de software y hardware. Puesto que las personas también forman parte del sistema, ciertas tareas del sistema podrían asignarse a seres humanos.
- *Requisitos funcionales (functional requirements)*: especifican los requisitos del software que se debe desarrollar para permitir a los usuarios realizar sus tareas,

satisfaciendo así los requisitos del sistema. Tradicionalmente estos requisitos se documentan como: "El sistema deberá…" (en inglés "The system shall...").

- *Reglas del negocio* (*business rules*, que en SIREN también se denominan *requisitos del dominio*): incluyen cuestiones transversales a las funciones del sistema, tales como políticas corporativas, regulaciones administrativas, estándares industriales, prácticas contables y algoritmos computacionales. Para muchos autores las reglas de negocio no son en sí mismas requisitos del software ya que existen fuera de los límites de cualquier sistema software específico. Sin embargo, las reglas de negocio obligan a que el sistema contenga funcionalidad que fuerce el cumplimiento de las mismas.
- *Requisitos no funcionales o de calidad (non-functional requirements)*: estos requisitos extienden la descripción de la funcionalidad del sistema o del software proporcionando características emergentes que son importantes para los usuarios y los desarrolladores. Por ejemplo, los requisitos no funcionales incluyen cuestiones como facilidad de uso, integridad, portabilidad, eficiencia y robustez. Otros requisitos no funcionales describen interfaces externas entre el sistema y el mundo exterior y restricciones sobre el diseño y la implementación. Las restricciones limitan las elecciones disponibles para el diseño y construcción del producto.
- *Características (features)*: Son abstracciones de conjuntos de requisitos funcionales relacionados que proporcionan una capacidad distinguible por parte del usuario y posibilitan la satisfacción de un requisito del negocio. Una característica está relacionada con un grupo de requisitos reconocible por un interesado, que ayuda en la toma de una decisión de compra del producto –en muchas ocasiones una característica se expresa tradicionalmente como un *ítem* distinto en una lista que describe el producto–.
- *Casos de uso (use cases)*: describen secuencias de interacciones entre el sistema y un actor externo (escenarios). Un actor es una persona, otro sistema software o un dispositivo hardware que interacciona con el sistema para conseguir un objetivo. Un caso de uso refleja una actividad independiente del negocio o del software que un actor puede iniciar con el objetivo de alcanzar un valor. Un caso de uso engloba generalmente distintos requisitos funcionales que sirven a un objetivo común.

Un requisito debería satisfacer las características descritas a continuación [296]:

- *Completo*: cada requisito debería describir completamente la funcionalidad que debe ser entregada. Debería contener toda la información necesaria para que el desarrollador diseñe e implemente dicha porción de funcionalidad. Si conscientemente se está omitiendo cierta información se recomienda usar el acrónimo TBD (*To Be Determined*) como una bandera estándar para resaltar este estado, que deberá ser resuelto.
- *Consistente*: un requisito consistente no debe tener conflictos con otros requisitos del mismo tipo o con requisitos de sistema o del negocio de más alto nivel.
- *Correcto*: cada requisito debería describir de forma precisa la funcionalidad que debe ser construida. La fuente del requisito (atributo *fuente*, ver Sección 2.6.2),

ya sea un usuario o un requisito de alto nivel del negocio o sistema, debe servir como referencia para la corrección.

- *Viable*: debe ser posible implementar cada requisito teniendo en cuenta las capacidades y limitaciones conocidas del proyecto, del sistema y de su entorno operativo.
- *Necesario*: cada requisito debe documentar una capacidad que los clientes necesitan realmente o que es requerida para conformar con un requisito de un sistema externo o de un estándar.
- *Priorizado*: se debe asignar una prioridad de implementación a cada requisito funcional, característica o caso de uso, para indicar cómo de esencial es para una determinada versión del producto.
- *No ambiguo*: todos los lectores de un requisito deberían llegar a una interpretación única y consistente del mismo. El lenguaje natural es altamente dado a la ambigüedad, y por tanto, los requisitos se deben escribir con un lenguaje simple, conciso y directo, apropiado al dominio del usuario.
- *Verificable*: se deben poder diseñar casos de prueba u otras aproximaciones de verificación (tales como inspección o demostración) para comprobar si el producto o sistema implementa adecuadamente cada requisito. Si un requisito no es verificable no se puede determinar de manera objetiva si fue correctamente implementado.
- *Modificable*: un requisito debe ser revisado cuando sea necesario y se debe mantener una historia de los cambios realizados sobre él. Esto obliga a que cada requisito sea etiquetado de forma única y especificado de manera separada de otros requisitos, de manera que se pueda referenciar de forma unívoca.
- *Trazable*: un requisito trazable puede ser enlazado hacia atrás, a su origen, y hacia delante a los elementos de diseño o código fuente que lo implementan y a los casos de prueba que verifican que la implementación es correcta.

Independientemente de los niveles de requisitos presentados anteriormente, en SIREN, cada requisito tiene asociado un *tipo de requisito*. Por ejemplo, requisitos de sistema (SYRS), características (FEAT), requisitos del software (SRS), requisitos de seguridad del software (SRSS), etc. El tipo de requisito determina el formato del identificador único del requisito y su conjunto de atributos. Cada catálogo de requisitos tiene un conjunto predefinido de tipos de requisitos, que puede ser ampliado. Cuando se reutiliza un requisito, es preciso tener en cuenta los atributos de los tipos del requisito origen y del requisito destino (Sección 2.6.2).

2.6.2 Guías para la definición de atributos

Cada tipo de requisito tiene predefinido un conjunto de atributos que permiten almacenar información acerca de distintas características de los requisitos del tipo. Para cada tipo de requisitos, el conjunto de atributos puede ser ampliado.

En SIREN, todos los tipos de requisitos tienen asociado un conjunto mínimo de atributos, que se denomina MIN [226, 280] y que se muestra a continuación. Están

marcados como *forzosos* aquellos a los que hay que asignar un valor obligatoriamente en el momento de la creación del requisito:

- *texto* (forzoso): la sentencia en lenguaje natural que especifica el requisito.
- *PUID (Project Unique IDentification) o identificadorÚnico* (forzoso) del requisito dentro del proyecto.
- *riesgo*: se debe ponderar cada uno de los requisitos comparándolos con el resto y realizar una estimación del riesgo inherente al requisito (por ejemplo: *alto*, *medio* y *bajo*).
- *criticidad*: cómo de importante es el requisito para el cliente. Por ejemplo, un esquema sencillo es prioridad *alta* –requisito obligatorio–, *media* –requisito recomendable–, *y baja* –requisito opcional–. En el caso de que exista una relación de exclusividad (Sección 2.6.4) entre dos o más requisitos obligatorios, se entenderá que sólo uno de ellos debe estar obligatoriamente en la especificación final de requisitos.
- *prioridad*: el valor de este atributo será establecido por el analista y ayudará a establecer un orden en el desarrollo. Los atributos *riesgo* y *criticidad* puede servir como guía para establecer el valor de la prioridad.
- *motivación*: indica la razón por la que el requisito está incluido en el proyecto.
- *estado*: se identifican nueve estados en los cuales un requisito se puede encontrar, tal y como se muestra en la Figura 7.
    - *Pendiente de definición*: la redacción del requisito reutilizado o extraído no se considerada definitiva. Por ejemplo, se ha reutilizado un requisito parametrizado que no ha sido instanciado todavía o se ha definido un requisito parametrizado al que todavía no se ha asignado un tipo al parámetro. En el caso de que el requisito esté incompleto, este estado también se puede indicar explícitamente en el texto del requisito utilizando el acrónimo TBD (*To Be Determined*).
    - *Pendiente de revisión:* el texto del requisito ya se considera completo y está en espera de su revisión (análisis y negociación).
    - *Pendiente de documentación:* tras el análisis y la negociación el requisito es aprobado y está pendiente de su documentación, es decir, de su ubicación definitiva en el documento de especificación de requisitos y, opcionalmente, de la revisión del texto y/o atributos del requisito.
    - *Descartado*: como resultado del análisis y negociación o bien de la validación el requisito es descartado y debe ser renegociado o eliminado.
    - *Pendiente de validación*: una vez que el requisito ha sido documentado se encuentra en estado pendiente de validación. Después de la validación, es posible que el requisito pase al estado *Descartado* si precisa su renegociación.
    - *Validado en análisis*: el requisito ha sido aceptado por todos los interesados, usuarios clave incluidos.
    - *Modelado*: el requisito ha sido modelado en análisis y/o diseño.
    - *Implementado*: el requisito ha sido codificado.

- *Verificado*: una vez implementado, se comprueba por parte del equipo de desarrollo que la implementación se ajusta a la especificación del requisito. En caso contrario el requisito requiere la revisión de su implementación
- *Validado en implementación*: una vez verificado, la implementación del requisito ha sido aceptada por todos los interesados, incluidos los usuarios clave. Si la implementación del requisito no es aceptada, el requisito puede requerir su redefinición o la revisión de la implementación.

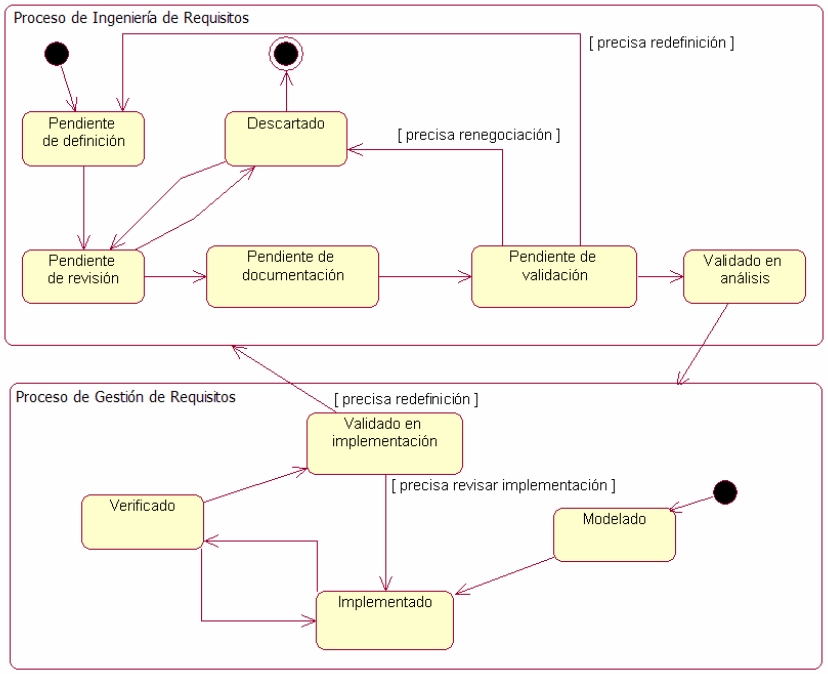

Figura 7 Diagrama de estados de un requisito

- *fuente:* es un campo de texto en el que se debe poner de dónde procede el requisito. Los requisitos pueden proceder (1) de las necesidades del cliente (y entonces en este atributo se indicará el nombre del usuario clave o típico involucrado); o (2) de restricciones sobre la solución técnica del proyecto, de la legislación vigente o de estándares (y entonces en *fuente* se deberán plasmar las referencias bibliográficas, direcciones web, estándares, etc. correspondientes). En un requisito que haya sido reutilizado del repositorio, este atributo *fuente* sirve también para reflejar de qué catálogo proviene y de qué requisito reutilizable.
- *criteriosValidación:* indica los criterios de validación que son necesarios para probar el requisito. Normalmente estos criterios se incluyen en el documento STS (*Software Test Specification*), descrito en la Sección 2.6.5.
- *propuestoPor:* indica el usuario que solicita que el requisito sea incluido.
- *responsable***:** miembro del equipo de desarrollo responsable de la implementación del requisito.
- *sección*: indica la sección del documento en la que está especificado el requisito. Por ejemplo, “1.2 Alcance”, “3.1.1 Interfaces de usuario”. En este atributo se incluye, además del número de la sección, el nombre de la misma para facilitar búsquedas de requisitos a reutilizar por este atributo.

- *históricoVersiones:* realiza un registro de todas las versiones del requisito: autor de la formulación o reformulación del requisito, fecha, versión y texto de la versión enésima

Cuando se reutiliza un requisito perteneciente a un tipo A, el nuevo requisito pertenecerá a un tipo B. Este tipo B podrá ser el mismo que A, en cuyo caso todos los valores de los atributos del requisito reutilizado serán los mismos que tenía el requisito original de A, excepto el campo *fuente* que indicará la procedencia del requisito reutilizable del que es copia. Sin embargo, si B es distinto de A, entonces el requisito reutilizado tendrá los mismos valores que el requisito reutilizable en los atributos pertenecientes al conjunto MIN, puesto que MIN $\subseteq$ A $\wedge$ B. El valor de un atributo de B que no esté en A será el valor por defecto del tipo de dicho atributo de B. La herramienta CARE de soporte a SIREN deberá permitir la modificación de los valores de los atributos propios de B (no incluidos en A) del requisito reutilizado.

2.6.3 Guías para la definición de requisitos parametrizados

Un requisito se ha de parametrizar si ciertas opciones se pueden elegir entre un conjunto de posibles valores cuando el cliente adquiere el producto o el equipo de desarrollo especifica los requisitos del sistema a desarrollar. Un *requisito parametrizado* dispone de uno o varios *parámetros* que sirven para diferenciar un producto reutilizado de otro posible producto reutilizado dentro del catálogo. El parámetro sirve para especificar un *punto de variación* en la especificación de requisitos, esto es, un punto en el que se debe seleccionar entre distintas alternativas para la configuración de un producto concreto (en el Capítulo 3 se introduce este concepto con más profundidad al presentar el concepto de variabilidad en líneas de productos). Si todos los tipos posibles deben ser soportados por el producto, y se puede elegir entre uno u otro en tiempo de explotación y no en tiempo de desarrollo, tal cuestión no se debería especificar como un parámetro. Por ejemplo:

> SRS1. El sistema permitirá la exportación a ficheros externos para la generación de informes.
> - De tipo CVS
> - De tipo Excel

El anterior sería un requisito parametrizado si cuando el cliente adquiere el producto se puede elegir entre una u otra clases de exportación (o ambas), quedando el requisito siguiente:

> SRS2. El sistema permitirá la exportación a ficheros externos para la generación de informes en [unFormato].

donde el valor del parámetro `unFormato` podría ser: CVS o Excel.

Si no fuera así, es decir, si las dos exportaciones debieran ser soportadas por el producto, no se debería especificar como un requisito parametrizado, sino más bien cómo tres requisitos diferentes con una *relación padre-hijo* (esta relación se describe en la Sección 2.6.4):

SRS1. El sistema permitirá la exportación a ficheros externos para la generación de informes en cualquiera de estos formatos:
SRS1.1 Formato CVS
SRS1.2 Formato Excel

Todo parámetro viene caracterizado por tres atributos:

- *nombre*.
- *tipo*: en todo catálogo existen dos tipos predefinidos: *numérico* y *cadena de texto*. Estos tipos se pueden ampliar con la definición de nuevos tipos específicos para cada catálogo. Los nuevos tipos representan un conjunto finito de valores (enumerados), y será necesario determinar el número mínimo y máximo de valores que el cliente podrá elegir en el momento de la instanciación. Si el mínimo y el máximo se han fijado con valor 1 quiere decir implícitamente que el tipo no es *multivaluado*.
- *alcance*: un parámetro puede ser *local* a un requisito o *global* al catálogo, en cuyo caso al tomar un valor todos los requisitos que incluyan ese parámetro tomarán el mismo valor.

A continuación se presentan ejemplos de requisitos parametrizados extraídos del catálogo SEGURIDAD-PDP (*Protección de Datos de carácter Personal*) (Sección 2.7.1):

SRS2. El responsable del fichero elegirá un [unSoporteFisico] para la realización de copias de seguridad.

Parámetro [nombre="unSoporteFisico", tipo =SOPORTE_FÍSICO b(1,1), *alcance* = GLOBAL]

SRS3. El responsable del fichero adquirirá la [unaUnidad], para el [unSoporteFisico] elegido.

Parámetro [*nombre = "unaUnidad", tipo=* UNIDAD_ALMACENAMIENTO {disco duro, disco duro externo, disquete, CD-ROM, DVD, unidad ZIP, unidad de cinta, pendrive, Blu-ray, HD-DVD, disco USB} (1,1), *alcance* = GLOBAL]

Vemos que el parámetro `unSoporteFisico` se ha definido con alcance global, por tanto, ambos requisitos (SR2 y SR3) deben tener el mismo valor para este parámetro cuando se instancien.

SRS4. El periodo máximo de almacenamiento de los pedidos será de [tiempoEnDias] días.

Parámetro [*nombre* = "*tiempoEnDias*", *tipo* = Numérico, *alcance* = LOCAL]

El requisito anterior también podría incluir un nuevo parámetro para determinar el periodo (días, meses, etc.) Si el conjunto de valores para los posibles periodos de tiempo es finito, se definiría un nuevo tipo de parámetro en el catálogo, por ejemplo, UNIDAD_TIEMPO.

SRS4bis. El periodo máximo de almacenamiento de los pedidos será de [tiempoEnDias] cada [unidadTiempo].

Parámetro [*nombre = "unidadTiempo"*, *tipo* = UNIDAD_TIEMPO{día, mes, año}(1,1), *alcance* = LOCAL]

SRS5. El sistema transformará datos de una fuente que estarán en [formato].

Parámetro [*nombre* = "formato", *tipo* = FORMATO{xml, jpg, texto}(1,3), *alcance* = LOCAL]

En este último ejemplo vemos que se trata de un parámetro multivaluado puesto que se puede elegir en el momento de la instanciación del parámetro de uno a tres valores.

Un requisito parametrizado inicialmente se encuentra en estado TBD, el valor del atributo *estado* es *Pendiente de definición*. Este valor no cambiará a *Definido* hasta que todos los parámetros que contenga en su especificación hayan sido instanciados.

En tiempo de reutilización no se pueden añadir nuevos valores a un parámetro. Si un parámetro p tiene como posibles valores a, b y c, el Analista no puede añadir un nuevo valor d, sino que debe dejar sin instanciar el parámetro p y utilizar después la aplicación de gestión de parametrizados de la herramienta (Sección 2.9) para añadir el nuevo valor.

2.6.4 Guías para la definición de trazas entre requisitos

Como se resume en la Sección 2.2.3.4, las trazas permiten establecer relaciones entre requisitos. El modelo de trazas definido en SIREN incluye las trazas *padre-hijo*, *requiere*, *relacionadoCon*, *exclusiva* y *materializaEn*. En las siguientes secciones se describen con detalle estas trazas.

2.6.4.1 Traza *padre-hijo*

Definición

Relación de descripción de un requisito más general por parte de una secuencia de requisitos específicos. Los identificadores de los hijos extienden el identificador del padre con la notación idPadre.1, idPadre.2, etc. En el documento y en el repositorio de requisitos, usualmente el texto del padre y el de los hijos está localizado espacialmente.

Excepción a la regla general de que los requisitos sean autocontenidos: se permite que el texto del padre esté escrito parcialmente, siendo completado por los hijos.

En tiempo de reutilización (desarrollo *con* reutilización):

(a) Los requisitos hijos son opcionales (el punto de variación viene dado implícitamente).

(b) Los requisitos hijos sólo tienen sentido en el contexto del padre: por tanto, reutilizar un hijo implica la reutilización obligatoria del padre y el mantenimiento de la relación *padre-hijo*.

Uso

Cuando un requisito se descompone directamente en otros:

(a) Descripción de los componentes, atributos o características de un objeto de negocio o de un componente de la interfaz. Por ejemplo: REQ1. Un socio es descrito por los siguientes campos: REQ1.1. Nombre. REQ1.2. DNI. REQ1.3. Domicilio. (…)

(b) Descripción de las distintas alternativas para desarrollar un proceso o un objeto. Por ejemplo: REQ1. La autenticación del usuario se puede realizar: REQ1.1. Por contraseña. REQ1.2. Por la voz. REQ1.3. Por el iris. Este tipo de requisitos no se debería especificar como parametrizado en el caso de que los hijos fueran susceptibles de tener una traza con otros requisitos: en ese caso el punto de variación siempre se tiene que especificar mediante una relación *padre-hijo*.

(c) Enumeración de ejemplos que sirven para mejorar la comprensión del requisito padre. Tras la reutilización, se debe asegurar que o bien se eliminan los ejemplos, o se amplían para cubrir todas las alternativas exigidas por la aplicación: un conjunto incompleto de ejemplos no se puede considerar propiamente un conjunto de requisitos.

(d) Especificación de casos de uso. El nombre del caso de uso se pondría en el texto del requisito, y los pasos de los escenarios se pondrían como requisitos hijos: propiamente "requisitos" serían únicamente los pasos realizados por el sistema, que deberían ser escritos como requisitos atómicos. Sin embargo, para que se pueda reutilizar el contexto de los requisitos, las acciones de los actores también se han de poner como requisitos. Los campos de la plantilla de descripción del caso de uso también serían requisitos hijos. Obsérvese que el hecho de especificar los casos de uso mediante relaciones *padre-hijo* y no otro tipo de relaciones implica que el grano de reutilización sea el del caso de uso, no el del paso del caso de uso: un hijo (paso) no se puede reutilizar sin su padre (caso de uso), y por tanto no se pueden reutilizar pasos individuales (si se considera esta limitación demasiado fuerte, habría que usar la traza *relacionadoCon* –ver más adelante).

(e) Para organizar los requisitos según el criterio usado para estructurar el repositorio (funcional, estímulo, escenario, estado, etc., ver modos de organización según el IEEE 830 en la Sección 2.6.8), no se usa *padre-hijo*: se usa la traza *relacionadoCon* –ver más adelante.

2.6.4.2 Traza *requiere*

Definición

Relación de dependencia direccional entre dos requisitos. A *requiere* B significa que B es una precondición de A. Ello implica que para que A se cumpla en el nuevo sistema B

se tiene que cumplir necesariamente. Por tanto, la reutilización de A implica obligatoriamente la reutilización de B. Se podría decir que *requiere* es una relación inclusiva fuerte.

Uso

La existencia de un requisito es condición necesaria para la existencia de otro. Por ejemplo, dados Req.A. El botón de alarma debe tener el icono de una campana y Req.B. El ascensor ha de proporcionar un botón de alarma, tenemos Req. B *requiere* Req.A.

2.6.4.3 Traza *relacionadoCon* (*Clúster de requisitos*)

Definición

Relación de dependencia bidireccional entre dos requisitos. Establece que un requisito A está "relacionado con" un requisito B: (1) A está relacionado con B, B refina o complementa A de alguna manera, de forma que cuando se reutilice A "se debe considerar" B para su posible reutilización; o (2) A y B forman parte del mismo *clúster* –ver uso (b) más abajo–. Se podría decir que *relacionadoCon* es una relación inclusiva débil.

Uso

(a) Enlace entre requisitos de distinto nivel de abstracción. Por ejemplo, un requisito de sistema *requiere* un requisito de software, un requisito de software *requiere* unas pruebas de aceptación. No son relaciones inclusivas fuertes (traza *requiere*) porque el diseño e implementación de un requisito se puede realizar de distintas maneras, y no se puede obligar a que sea de una forma determinada.

(b) Permite aumentar el tamaño del grano de la reutilización a través de la definición de *clusters de requisitos verticales*: descripción de una función, estado, etc. (según la organización de IEEE 830 escogida, ver Sección 2.6.8), de forma que los requisitos refinados sean tenidos en cuenta en tiempo de reutilización. La especificación de requisitos queda formada así por un conjunto de árboles de requisitos, cada uno de ellos jerarquizado por función/subfunción, estado/subestado, etc.

(c) Ayuda a la definición en la especificación de requisitos de *clusters de requisitos horizontales* o *aspectos de requisitos*: a partir de cuestiones como la seguridad o la protección de datos personales, las relaciones inclusivas débiles podrían ser "transversales", horizontales a los árboles citados en el punto (b) anterior, atravesando transversalmente la jerarquía de documentos de requisitos y la organización escogida de tales documentos. Examinando el apartado del documento al que pertenece el requisito raíz de estos árboles horizontales se podría conocer el clúster al que pertenece un requisito (seguridad, por ejemplo), pero parece adecuado introducir un nuevo campo en los requisitos de SIREN, *clúster*, que recoja una colección de las "etiquetas" que se han asignado a las trazas entre requisitos: [seguridad], [protección de datos], etc. Como vemos, los clusters no son disjuntos, pues se trata de una colección de etiquetas para las relaciones entre requisitos. Un

requisito puede así estar relacionado con distintos requisitos de acuerdo a distintos criterios (clusters). La relación *padre-hijo* no se puede usar para definir estos clusters ya que un requisito no puede tener más de un padre.

2.6.4.4 Traza exclusiva

Definición

Relación de exclusividad entre dos requisitos: R1 exclusivo-con R2 significa que si R1 está presente en la especificación R2 no puede estar, y recíprocamente si R2 está en la especificación R1 no puede estar.

Uso

(a) Dos requisitos están en conflicto y no se pueden satisfacer simultáneamente.

(b) Con los requisitos exclusivos el punto de variación queda implícito: se resuelve al escoger un requisito u otro. Si dicho punto de variación se refiriera a más de dos requisitos, la especificación se puede volver engorrosa, al tener que especificar relaciones de traza exclusiva dos a dos entre todos los requisitos implicados. Puede ser preferible usar en ese caso una relación *padre-hijo*, indicando en el texto del padre que los hijos son exclusivos.

(c) Requisitos parametrizados en los que hay relación de trazas entre las distintas opciones del parámetro: la solución es hacer explícita cada opción como un requisito distinto, manteniendo relaciones de exclusividad entre ellos. Pero como se comenta arriba en (b), si la especificación se hace engorrosa tal vez sea preferible usar relaciones *padre-hijo*.

2.6.4.5 Traza *materializaEn* (*reifies*)

Definición

Relación entre un requisito y un artefacto del desarrollo, como una clase, un módulo, un componente, etc.

Uso

Ligadura entre un requisito y un artefacto del desarrollo, representado por un tipo de requisito (clase, módulo, componente, etc.).

2.6.5 Guías para la organización de los catálogos

En SIREN el *repositorio* de requisitos reutilizables está formado por un conjunto de *catálogos* de requisitos reutilizables que pueden ser *dominios* o *perfiles* (ver Sección 2.4). A su vez, cada uno de estos catálogos se organiza a partir de la jerarquía de documentos de requisitos que aparece representada en Figura 8. Cada documento se corresponde con un nivel de especificación diferente y por tanto, tiene objetivos y destinatarios distintos.

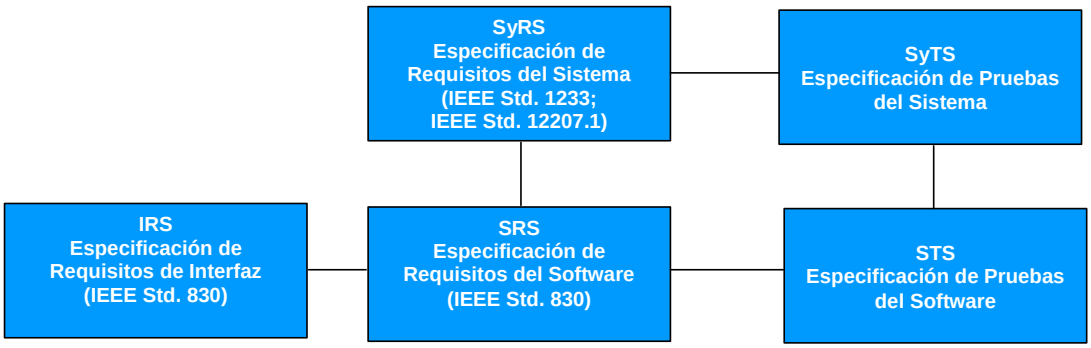

Figura 8 Una posible jerarquía de documentos de requisitos para SIREN

En la Figura 8 aparecen cinco tipos distintos de documentos. No son todos obligatorios para cualquier catálogo o proyecto. La decisión sobre la jerarquía a utilizar en el proyecto actual corresponde al jefe de proyecto junto a los analistas (ver Tabla 12), y depende del tamaño del proyecto y de la complejidad del sistema que se va a desarrollar. Cuando en esta memoria de tesis doctoral hablamos genéricamente de *documento de requisitos* se hace referencia bien a un documento concreto de la jerarquía o bien a la jerarquía completa. Los documentos considerados en la Figura 8 se describen a continuación:

- Los documentos de Especificación de Requisitos del Sistema (que se referenciarán por sus siglas en inglés, *System Requirements Specification*, *SyRS*) y Especificación de Pruebas del Sistema (*System Test Specification*, *SyTS*) son únicos y opcionales. No son necesarios cuando el problema es simple y está claro desde el principio que todos los requisitos van a ser soportados por el software –por ejemplo, en el desarrollo de un sistema autónomo simple y bien conocido–.
- El documento de Especificación de Requisitos del Software (*Software Requirements Specification*, *SRS*) es obligatorio y puede dividirse en subdocumentos (por ejemplo, un SRS por cada módulo o subsistema).
- El documento de Especificación de Requisitos de Interfaz (*Interface Requirements Specification*, *IRS*) es único y opcional y se utiliza cuando los requisitos de interfaz son muy numerosos, con el objeto de evitar que el SRS sea muy extenso; en otro caso, estos requisitos se incluyen en el SRS.
- Se debe definir un documento de Especificación de Pruebas del Software (*Software Test Specification*, *STS*) por cada SRS de la jerarquía.

En SIREN las plantillas de documentos de requisitos están basadas en los estándares de IEEE (Figura 8). Sin embargo, en la práctica los documentos de requisitos raramente se derivan directamente del estándar correspondiente, sino que cada organización debe particularizarlos teniendo en cuenta el tipo de proyectos que desarrolla y el tamaño y la cultura del equipo de desarrollo. A continuación se describen estas plantillas:

- *Especificación de Requisitos del Sistema (SyRS).* De acuerdo con el estándar IEEE 12207.0 [140], el SyRS debe incluir: (1) funciones y capacidades del sistema (requisitos del sistema); (2) requisitos del negocio, organizativos y de usuario; (3) requisitos de seguridad (*security*), seguridad a terceros (*safety*) y privacidad (*privacy*); (4) requisitos de ingeniería de factores humanos; (5)

requisitos de operaciones y mantenimiento; y (6) restricciones de diseño. A partir de estas cuestiones, el estándar IEEE 12207.1 [141] detalla el contenido específico de la plantilla del SyRS haciendo referencia al estándar IEEE 1233 [139], *Guide for Developing System Requirements Specifications*, donde se pueden encontrar guías para especificar los requisitos del sistema y la propuesta de un esquema del SyRS. Este esquema es la base de la plantilla del SyRS de SIREN.

- *Especificación de Requisitos del Software (SRS).* La mayoría de los requisitos del software se obtienen directamente a partir de los requisitos del sistema; de ahí la visión jerárquica presentada en la Figura 8. En SIREN el SRS se basa en el estándar IEEE 830 [138], en el que se recogen requisitos acerca de la funcionalidad del sistema (requisitos funcionales); interfaces externas; restricciones de diseño; requisitos no funcionales o de calidad (rendimiento, portabilidad, mantenimiento, seguridad, disponibilidad y fiabilidad); características (*features*) y reglas de negocio.
- *Especificación de pruebas del sistema y del software (SyTS y STS).* En los documentos de pruebas (SyTS o STS) se especificarán casos de prueba para asegurar que el sistema o el software cumple los requisitos especificados en el SyRS y SRS.
- *Especificación de requisitos de interfaz (IRS).* Los requisitos relacionados con los interfaces entre los elementos del software y entre el usuario y el sistema podrían incluirse en el SRS. No obstante, con el objetivo de reducir el tamaño del SRS en el caso de documentos muy extensos, en algunas ocasiones es conveniente hacer uso de un documento separado del SRS denominado IRS. Por tanto, se deben establecer las relaciones de traza adecuadas entre el SyRS, el SRS y las interfaces descritas en el IRS. La plantilla del IRS tiene la misma estructura que el apartado del SRS dedicado a la especificación de requisitos de interfaz [138].

Si bien esta es la organización documental propuesta en SIREN para la especificación de requisitos, usualmente las herramientas CARE almacenan los requisitos en una base de datos, de modo que los analistas pueden trabajar directamente sobre esta base de datos, en lugar de manejar documentos textuales de requisitos. Por tanto, también se debe prestar atención a la organización de los requisitos en la base de datos. Es recomendable organizar los requisitos en la base de datos en carpetas o módulos que mimeticen la estructura de los documentos de requisitos, en lugar de tener una larga lista no estructurada de requisitos.

2.6.6 Guías de reutilización de requisitos

Una vez establecido el ámbito de alcance del proyecto se debe buscar en el repositorio SIREN si existe algún catálogo de dominio conforme a dicho ámbito. Si es así, posiblemente el proyecto se corresponde con el desarrollo de un producto concreto del dominio especificado en el catálogo. En este caso, las búsquedas en el repositorio podrían comenzar buscando en el catálogo del dominio los requisitos con el valor *alta* en el atributo *criticidad* (Sección 2.6.2). Al ser obligatorios, estos requisitos forman parte de cualquier producto de dicho dominio y por tanto deben reutilizarse todos ellos,

analizando sus trazas para determinar otros requisitos no obligatorios que también deban formar parte de la especificación del proyecto actual.

Una vez establecida la especificación común del producto se deben definir nuevas búsquedas guiadas por los requisitos del negocio y las características (*features*) identificadas. Estas guías pueden implicar búsquedas en el mismo catálogo de dominio o bien en distintos catálogos de perfil.

Si como resultado de alguna de las búsquedas se reutiliza un requisito que tiene relaciones de traza *relacionadoCon* otros requisitos, se ha encontrado un clúster de requisitos. En este caso se debe considerar la selección de los requisitos englobados en dicho clúster.

La reutilización de los requisitos seleccionados implica la resolución de los puntos de variación encontrados en tales requisitos:

- Instanciación de los parámetros de los requisitos parametrizados con los valores adecuados al proyecto actual.
- Resolución de trazas exclusivas.
- Resolución de trazas *relacionadoCon*, que son opcionales.
- Resolución de trazas *requiere*, que deberían incluirse en condiciones normales.
- Resolución de trazas *padre-hijo*, que deberían incluirse en condiciones normales.

2.6.7 Guías para la mejora del repositorio

Se recomienda crear una línea base (*baseline*) para cada catálogo del repositorio cuando finaliza un proyecto que lo haya utilizado y haya implicado una mejora del mismo.

Un proceso sencillo de control de cambios del repositorio es el siguiente:

1. Los analistas que estén trabajando en un proyecto en el que se reutilicen requisitos de algunos de los catálogos, podrán enviar peticiones de cambio al responsable del catálogo (Tabla 12). Además de las peticiones de cambio de requisitos existentes en el catálogo, también pueden formular peticiones para la inclusión de nuevos elementos en el catálogo (tipos de parámetros, requisitos del proyecto actual, etc.) tras la validación de los mismos.
2. El equipo responsable del catálogo analizará las peticiones de cambio para determinar su viabilidad. En el caso de que la petición implique una mejora en un catálogo (cambios menores en la redacción de un requisito, cambios en los valores de los atributos, etc.), el cambio podría ser automáticamente incorporado y con ello visible a todos los proyectos que estén utilizando el catálogo. En el caso de que la petición de cambio implique una extensión o reducción de los requisitos del catálogo (ampliación o reducción de los valores de un tipo parametrizado, nuevas trazas exclusivas, etc.), es preciso estudiar los proyectos en marcha que hacen uso del catálogo para determinar cómo les afecta el cambio.

2.6.8 Guías para la organización de la documentación de especificación de requisitos

De manera general, la organización del documento de especificación de requisitos de un producto o sistema concreto se puede abordar de acuerdo con las plantillas propuestas por el estándar IEEE 830 [138]. En la Sección 3 de la estructura propuesta por el estándar se proponen distintas organizaciones para los requisitos específicos del sistema: (1) por modos; (2) por clases de usuarios; (3) por objetos; (4) por características (*features*); (5) por estímulos; y (6) por jerarquía funcional. Una u otra organización se debe elegir según el ámbito del proyecto y la cultura del equipo de desarrollo.

De forma más concreta unas guías sencillas para la obtención y refinamiento de los requisitos del proyecto son las siguientes:

- Realizar la declaración de ámbito de alcance.
- Establecer formalmente los límites del sistema (por ejemplo, mediante un diagrama de contexto [298]).
- Determinar los módulos de análisis.
- Determinar (reutilizar) las características (*features*) del sistema o producto a desarrollar.
- Determinar (reutilizar) los requisitos funcionales y no funcionales a partir de las características anteriores.
- Determinar (reutilizar) las reglas de negocio a partir del conocimiento de los dominios y perfiles implicados, incluyéndolas en un apartado diferenciado del documento de especificación de requisitos denominado “Reglas de negocio” o “Requisitos del dominio”.
- Recorrer los clusters de requisitos especificados en los catálogos reutilizables considerados en el proyecto para determinar requisitos funcionales, no funcionales y/o reglas de negocio que pueden ser reutilizados.
- Asignar las características, requisitos funcionales, no funcionales y reglas de negocio a los apartados del documento siguiendo el criterio de organización escogido.

Estos pasos se realizan con continuas iteraciones y solapamientos entre ellos.

2.6.9 Modelo de referencia inicial de SIREN

Los elementos que se han ido desarrollando a lo largo de la presentación de SIREN conforman el que denominamos “modelo de referencia del repositorio de requisitos reutilizables de SIREN”. Con el objetivo de resumir los elementos que se han desarrollado en la presentación de SIREN, en la Figura 9 se muestra una visión inicial de este modelo de referencia, que para facilitar su lectura no contempla todos los detalles.

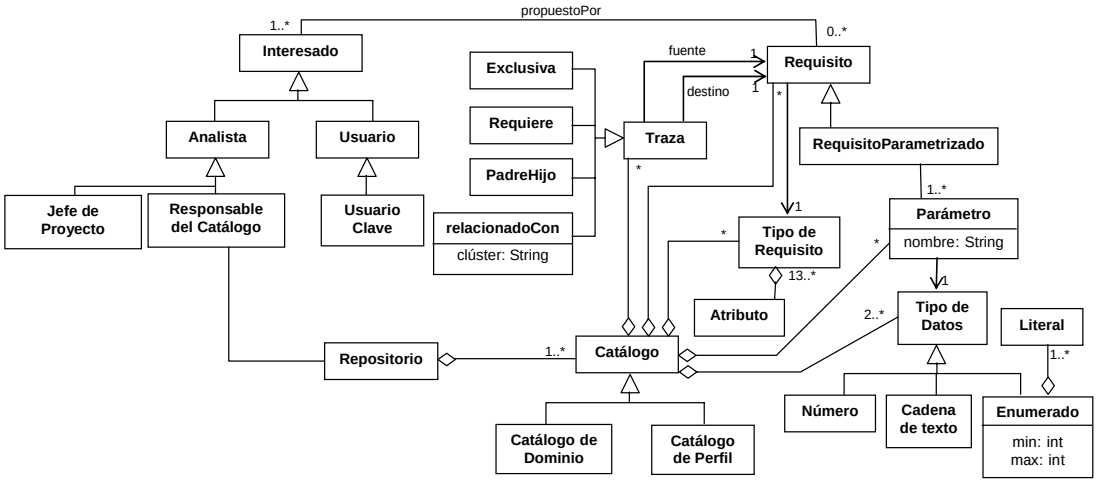

Figura 9 Modelo de referencia del repositorio de requisitos reutilizables de SIREN (versión inicial)

Se puede observar como el concepto de punto de variación no se refleja explícitamente en el modelo sencillo de la Figura 9. Como se ha mostrado en la presentación del método SIREN, los puntos de variación en el modelo de requisitos de SIREN vienen dados por:

- Requisitos parametrizados.
- Traza *exclusiva*.
- Traza *padre-hijo* en la que el texto del padre indica que algunos hijos son opcionales.
- Traza *relacionadoCon,* que no implica necesariamente la reutilización de los requisitos trazados.

En el Capítulo 4 de esta memoria de tesis doctoral se profundiza en este modelo de referencia con el objetivo de que sirva de soporte en la generación de requisitos textuales a partir de los modelos del dominio de SIRENspl planteados en el Capítulo 3.

2.7 Experiencias de aplicación de SIREN

En este apartado se repasan las experiencias en las que SIREN se ha aplicado en proyectos reales, tanto en su vertiente *para* reutilización como en su vertiente *con* reutilización. De acuerdo con la terminología definida en la Sección 1.3.1, en ocasiones se trata de "experiencias" (Sección 2.7.1) y en ocasiones de verdaderos "casos de estudio" (secciones 2.7.2 y 2.7.3), pues a la experiencia se une el hecho de haber definido previamente un conjunto de variables con vistas a ponderar los resultados no sólo cualitativamente sino también cuantitativamente.

2.7.1 Catálogos de requisitos reutilizables de seguridad

Actualmente hay definidos tres catálogos de requisitos reutilizables, que están a disposición de los usuarios de SirenTool (Sección 2.9), los tres en el ámbito de la

seguridad de sistemas de información. Según la terminología de SIREN son *perfiles*, pues tratan con dominios horizontales. Los dos primeros son los siguientes:

- Catálogo *Seguridad-PDP* (*Protección de Datos Personales*) [285], que incluye los requisitos procedentes de la Ley Orgánica de Protección de Datos Personales (LOPD) [184] (que es una adaptación de la directiva europea 95/46/CE) y del Reglamento de Medidas de Seguridad (RMS) [263].
- Catálogo *Seguridad-MAGERIT* [284], que incorpora requisitos procedentes de MAGERIT [202], el Método de análisis y gestión de riesgos de la administración pública española, basado en ISO/IEC 15408-1999, *Evaluation Criteria for Information Technology Security Standard*, también conocido como *Common Criteria Framework*–CCF.

En una primera fase, este doctorando ha participado en los trabajos iniciales con estos catálogos, especialmente con el de Seguridad-MAGERIT (compatible con MAGERIT 1.0) [281, 284], y ha participado activamente en su revisión en el marco del proyecto fin de carrera de Miguel Ángel Martínez Aguilar [205]. Con posterioridad, en una segunda fase, el doctorando Joaquín Lasheras se ha especializado en esta línea de investigación en el marco de su tesis doctoral, ha generado las versiones actuales de los catálogos (revisándolas de nuevo y adaptándolas a MAGERIT 2.0), y ha desarrollado un nuevo catálogo, también disponible en SirenTool:

- Catálogo *Seguridad-ISO 27001*, que integra los requisitos de los catálogos anteriores y los relaciona con los objetivos del estándar de seguridad ISO 27001 [148] para un sistema de gestión de seguridad de la información.

Siguiendo a Toval *et al.* [281], en este apartado sintetizamos los antecedentes, objetivos y principales resultados de los primeros trabajos con los catálogos Seguridad-PDP y Seguridad-MAGERIT en los cuales este doctorando ha participado.

2.7.1.1 Antecedentes

En el campo de la seguridad se puede distinguir entre el concepto de *seguridad* propiamente dicho (en inglés, *security*: la capacidad de un sistema para gestionar, proteger y distribuir información sensible) y lo que llamaremos *seguridad respecto a terceros* (en inglés, *safety*: la ausencia de consecuencias catastróficas para el entorno) [4]. En los últimos años ha habido un interés creciente en la seguridad de los sistemas de información (en cuanto a *safety* y *security*), que están acechados por múltiples amenazas de distinta naturaleza. Por citar sólo algunas, (1) la falta de adecuación a la legislación vigente (como a la LOPD, que en nuestro país puede acarrear importantes sanciones administrativas, o multas de hasta €750.000); (2) la inexistencia de áreas de protección contra incendios; (3) la ausencia de SAIs (Sistemas de Alimentación Ininterrumpida) en los servidores; e incluso (4) se puede considerar como amenaza la huelga o marcha del personal estratégico de la organización. Es evidente que la aceptación generalizada del comercio electrónico y de la firma electrónica para transacciones administrativas y/o comerciales requiere un aumento de la confianza en la seguridad de los sistemas de información.

Esta situación ha promovido la aparición de numerosas normas y métodos relacionados con la seguridad informática, entre las que destacamos el ISO 15408, *Common Criteria*

Framework (CCF) [150], y los métodos CRAMM (en el Reino Unido), MARION (en Francia) y MAGERIT [200, 202], la Metodología de análisis y gestión de riesgos de la administración pública española, que es una adaptación de CCF. Aunque la necesidad de seguridad afecta a todas las formas de información y a sus soportes o bien a cualquier método usado para transmitir conocimiento, datos e ideas, MAGERIT se limita a considerar la seguridad de la información de los sistemas de información que se incluye en bases de datos, cintas o disquetes, dejando al margen flujos de información informales como por ejemplo las conversaciones.

La práctica habitual en las primeras etapas del desarrollo de sistemas de información ha prestado poca atención a la seguridad, que se ha considerado, en el mejor de los casos, en el diseño, implementación o pruebas, e incluso a veces sólo cuando el sistema ha sido totalmente construido. Sin embargo, la seguridad es barata a corto plazo: cuanto más temprano se actúe para dar seguridad a un sistema de información, más sencilla y económica resultará la seguridad en la organización [54, 200].

En este apartado presentamos una experiencia para considerar la seguridad de un sistema de información desde el proceso de IR, presentando un enfoque que se basa en la reutilización de requisitos procedentes de dos catálogos de requisitos de seguridad: (1) Seguridad-MAGERIT, procedente de MAGERIT 1.0 (y por tanto compatible con un subconjunto de CCF); y (2) Seguridad-PDP, procedente de la LOPD y el RMS. Hablamos de “experiencia” y no de “caso de estudio” pues en el proceso de creación de los catálogos no se realizó la recogida sistemática de unas variables definidas con antelación. Con este enfoque la elección de un requisito de seguridad implica la eliminación de un riesgo en el futuro sistema, garantizándose que un sistema de información que incorpore nuestros requisitos de seguridad superará con éxito un estudio de riesgos realizado con MAGERIT (que incorpora la LOPD).

Este enfoque basado en los catálogos de seguridad surge a partir de un contrato del GIS con la CARM (Comunidad Autónoma de la Región de Murcia), denominado proyecto CARMMA [1], con el objetivo de realizar un análisis de riesgos en la Dirección General de Sistemas de Información y Telecomunicaciones. Dos de las principales conclusiones del estudio fueron las siguientes: (1) el costo de implementar las salvaguardas para reducir el riesgo de la mayor parte de los activos críticos identificados habría sido menor si estos activos se hubieran desarrollado desde el principio teniendo en cuenta cuestiones de seguridad; (2) aunque las salvaguardas en MAGERIT están ligadas a amenazas, la implementación de cada salvaguarda depende del activo específico amenazado, y por tanto las salvaguardas deberían estar ligadas a activos. Estas dos conclusiones nos llevaron a la definición de un catálogo de seguridad, Seguridad-MAGERIT, estructurado de acuerdo con la jerarquía de activos de MAGERIT. Este catálogo ayuda a hacer explícitas las cuestiones de seguridad desde los pasos iniciales en el proceso de desarrollo del sistema. Para profundizar en las cuestiones relativas a la privacidad, el desarrollo de este catálogo fue seguido por el de otro catálogo relacionado con la protección de datos personales, Seguridad-PDP.

2.7.1.2 La seguridad y la ingeniería de requisitos

Como hemos dicho, la motivación esencial de esta propuesta es la incorporación explícita de la seguridad en el desarrollo de sistemas de información, desde el proceso de IR, de forma que se mejore la calidad. En este sentido, Sommerville [276] afirma que

un buen número de funcionamientos anómalos relacionados con la seguridad del sistema se deben a errores de especificación más que a errores de diseño. Por ejemplo, la especificación de requisitos puede ser incompleta y no describir la conducta del sistema en ciertas situaciones críticas. Por otro lado, en un estudio sobre sistemas empotrados, Lutz [188] concluye que "...la causa principal de los errores del software relacionados con la seguridad a terceros que han persistido hasta la integración y prueba del sistema son problemas con los requisitos".

Existe un número incipiente de iniciativas para incluir la seguridad dentro del ciclo de vida. Este número se reduce si consideramos sólo las iniciativas que incluyen la seguridad en el proceso de IR. Notables excepciones son la aproximación de Chung [71] y la integración de SSADM con CRAMM y de Métrica 3 con MAGERIT 1.0. Estas integraciones ligan expresamente el desarrollo del sistema de información (incluyendo la especificación de requisitos) con la seguridad, aunque no proporcionan nada parecido a un repositorio con requisitos reutilizables de seguridad.

Los estándares de seguridad tampoco soportan adecuadamente el proceso de IR. Creemos que parte del estudio de Fenton y Neil [100] sobre las carencias de los estándares de seguridad a terceros se puede aplicar también a los estándares de seguridad (en particular a ISO 15408): (1) los requisitos que imponen los estándares son imprecisos; y (2) los estándares son demasiado complejos y difíciles de utilizar. De hecho, en dicho estudio se propone un proceso de mejora de los estándares de seguridad a través de la mejora en la organización y escritura de los requisitos que contienen. Además, Fenton y Neil afirman que los estándares de seguridad a terceros están concebidos para evaluar sistemas ya creados, y no hacen hincapié en cómo dotar de seguridad a sistemas de nueva creación. En esta línea, pero en el campo de la protección de datos personales, Antón afirma que los requisitos de seguridad y privacidad han de estar alineados con las leyes y regulaciones existentes, que sin embargo describen los derechos y obligaciones en un lenguaje legal complejo y algunas veces ambiguo [45]. Esta autora ha realizado un extenso trabajo en el campo de los requisitos de protección de datos personales [26], incluyendo la definición de un método para extraer requisitos a partir de regulaciones legales [45].

En otra línea de trabajo, Mellado *et al.* [213] incorporan la seguridad en el paradigma de líneas de productos desde el proceso de requisitos, para lo cual proponen un modelo de variabilidad de seguridad y un modelo de decisión dirigido por estándares de seguridad como ISO/IEC 27001 [148] e ISO/IEC 15408 [150].

Creemos que todos estos trabajos respaldan el interés de desarrollar sendos catálogos de requisitos de seguridad reutilizables, compatibles con MAGERIT por un lado y con la LOPD y el RMS por otro, de manera que SIREN se pueda aplicar en el campo de la seguridad de los sistemas de información.

2.7.1.3 MAGERIT, la LOPD y el RMS como fuentes de requisitos de seguridad

MAGERIT [200, 202] es la Metodología de análisis y gestión de riesgos de sistemas de información de la administración pública española, compatible con ISO 15408, *Common Criteria Framework*. Para MAGERIT, el objetivo de seguridad de los sistemas de información consiste principalmente en mantener la continuidad de los procesos de la

organización que soportan dichos sistemas de información. Asimismo, se intenta minimizar tanto el coste global de la ejecución de dichos procesos como las pérdidas de los recursos asignados a su funcionamiento.

A principios de los 70, surgió en EE UU la primera generación de métodos para analizar los riesgos de los sistemas de información, básicamente listas de comprobación y modelos muy rudimentarios causa-efecto. A partir de mediados de los 80 surgió una segunda generación de métodos, más formalizados, encabezados por la británica CRAMM (*CCTA Risk Analysis and Management Method*). Desde finales de los 90 ha comenzado una tercera generación de métodos más relacionados con la legislación y normativa en materia de seguridad y más ligados a los métodos de desarrollo de sistemas de información. Es el caso de la nueva versión de CRAMM con SSADM o de la primera versión de MAGERIT con Métrica 2.1 y 3. En estos métodos de tercera generación se consideran los problemas organizativos de modo que se reconcilian la seguridad y funcionalidad del sistema, ya que se tienen en cuenta los aspectos de seguridad desde las primeras fases del proceso de desarrollo [33].

MAGERIT se dirige principalmente a sistemas ya existentes, mientras que Métrica es un método de desarrollo de nuevos sistemas de información. Para integrar ambos métodos, Métrica 3 aporta un *interfaz de seguridad* que indica las actividades a realizar para considerar la seguridad en todo el desarrollo.

En realidad, MAGERIT sólo se aplica a una de las fases de la gestión de la seguridad en los sistemas de información: el análisis y gestión de riesgos. A partir de los objetivos, estrategia y política de seguridad de los sistemas de información existentes en la organización, mediante la aplicación de MAGERIT se estudian primero los riesgos que soporta cada sistema de información y su entorno, y después se recomiendan las medidas (*salvaguardas*) que deberían ser adoptadas para registrar, prevenir, impedir, reducir o controlar los riesgos investigados. Estas salvaguardas quedan plasmadas en un *Plan de seguridad de los sistemas de información*, que después es implantado y cuya aplicación se somete a un seguimiento durante el mantenimiento. La aplicación de MAGERIT es cíclica: en la fase de mantenimiento se realizan periódicamente nuevos análisis de riesgos y se proponen nuevas medidas de salvaguarda.

El modelo de análisis y gestión de riesgos de MAGERIT 1.0 ofrece una guía sólida para la obtención de requisitos reutilizables de seguridad para nuevos proyectos. Este modelo incluye: (1) el *submodelo de elementos*, que proporciona las entidades básicas relacionadas con el análisis de riesgos del sistema de información: *activos, amenazas, vulnerabilidades, impactos, riesgos* y *salvaguardas*; y (2) el *submodelo de procesos*, que describe las etapas del proyecto de seguridad que se va a construir: *Planificación, Análisis de riesgos, Gestión de riesgos* y *Selección de salvaguardas.*

Los activos son los recursos del sistema de información o relacionados con éste (formando parte de su entorno). Como vemos en la Figura 10, MAGERIT 1.0 organiza los activos en cinco grandes capas o niveles, según sean más o menos tangibles (en MAGERIT 2.0 esta organización de los activos ha cambiado y se corresponde con lo descrito en el “Catálogo de elementos” [202]).

Figura 10 Jerarquía de activos de MAGERIT 1.0

Cada activo se caracteriza por su estado de seguridad, que se concreta estimando los niveles de cuatro estados de autenticación, confidencialidad, integridad y disponibilidad (ACID). Actualmente se tiende a considerar también la auditabilidad y el no repudio (ACIDAN).

La estructura del catálogo de requisitos reutilizables Seguridad-MAGERIT preserva la estructura de los activos de MAGERIT 1.0: (1) en la *Capa de entorno del sistema de información* se incluyen las instalaciones donde descansa el sistema de información, el mobiliario, los suministros, etc.; (2) la *Capa del sistema de información* (que se detalla en la Figura 11) se estructura en tres bloques, que se refinan a su vez: software, hardware y personal; (3) la *Capa de información* incluye la información manejada por el sistema de información: datos de las aplicaciones (incluyendo los informatizados y los resultantes del sistema de información), metadatos (como estructuras de datos o diccionarios de datos), y soportes (tratables informáticamente y no tratables). Es por tanto este nivel el que más interés tiene en relación con la seguridad en bases de datos; (4) la *Capa de funciones de la organización* justifica la existencia de los sistemas de información, dotándoles de finalidad: bienes y servicios producidos, y personal usuario de los servicios; y finalmente (5) la *Capa de otros activos* describe activos que frecuentemente son intangibles y no encajan en las categorías inferiores, como la credibilidad de la organización.

En la Figura 11 cada nivel se refina en *bloques* de activos homogéneos y cada bloque se divide a su vez en bloques más específicos. Las iniciales en la figura (IS, ISHW, ISSW, etc.) se usan para mantener la estructura de la jerarquía de activos de MAGERIT en el catálogo Seguridad-MAGERIT.

En la Capa 3 de MAGERIT 1.0 se menciona la protección de datos de carácter personal. Dada la importancia de la protección de datos, los requisitos relacionados con ésta, aun estando muy relacionados con los de seguridad, se consideran dentro del repositorio SIREN como un perfil aparte.

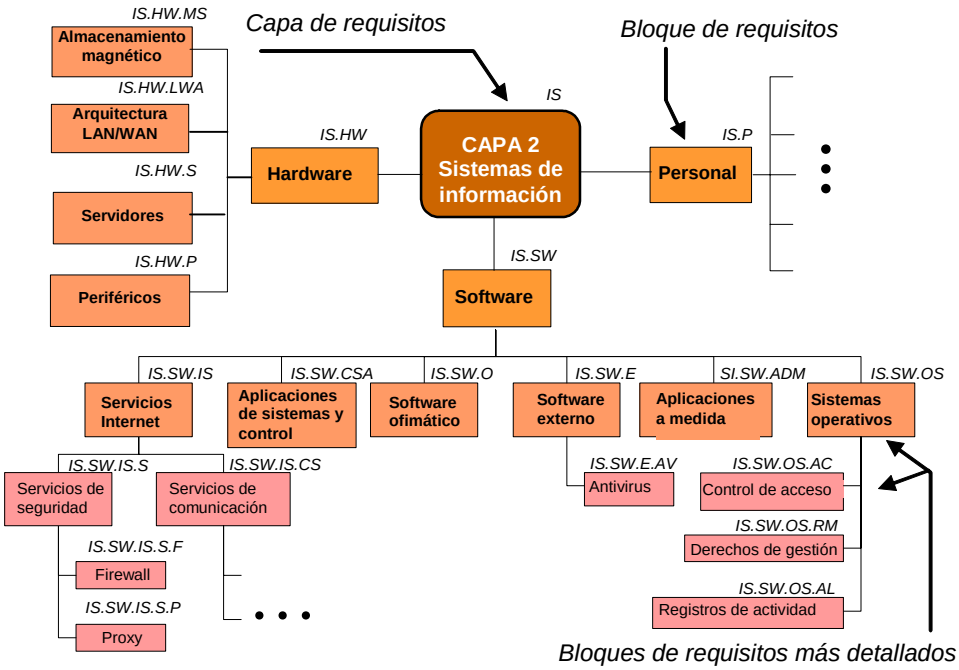

Figura 11 Un extracto de la Capa 2 de Sistemas de información de MAGERIT 1.0

2.7.1.4 Un ejemplo del catálogo Seguridad-MAGERIT

En la Figura 12 se muestran ejemplos extraídos del catálogo de Seguridad-MAGERIT. En dicha figura las etiquetas R1-R4 sólo se utilizan como abreviatura.

Los requisitos en el perfil de seguridad y de la LOPD redefinen el atributo *fuente* y *criticidad* de MIN, el conjunto mínimo de atributos de SIREN:

- *fuente*: en el caso de MAGERIT, el valor de este atributo se corresponde con la posición específica de cada requisito de seguridad en la jerarquía de activos de MAGERIT, esto es, el bloque y la capa (por ejemplo, acrónimos IP, IIGS, IPR, etc. en la Figura 11). De esta manera, el perfil de seguridad dentro del repositorio podría ordenarse de acuerdo a dos criterios: (1) el estándar para la especificación de requisitos; y (2) la estructura de MAGERIT. En el caso de la PDP, este atributo almacena la procedencia del requisito, sea la LOPD o el RMS, y el artículo correspondiente.
- *criticidad*: Proporciona al analista información sobre la necesidad de incluir los requisitos en un proyecto. En el caso de MAGERIT, los posibles valores para este atributo son: *obligatorio*, *recomendable* y *opcional*. La reutilización de los requisitos recomendables y opcionales dependerá del estudio de los riesgos que afectan al sistema de información, más o menos pormenorizado en función del proyecto. En el caso de la PDP, los posibles valores son: *obligatorio* y *recomendable*.

SyRS *(Especificación de Requisitos del Sistema)*

3. Capacidades del sistema. Condiciones y restricciones.

3.1. Físicas.

3.1.1. Construcción.

R1 (SYRS311S68) Los documentos y disquetes se guardarán bajo llave cuando no se estén utilizando y fuera de la jornada laboral.

3.3. Seguridad del sistema

3.3.1 Requisitos del sistema para alcanzar el nivel básico de seguridad

SyRS331L4. El procedimiento de notificación y respuesta ante incidencias contendrá un registro en el que se haga constar:

a) El tipo de incidencia.

b) El momento en que se ha producido la incidencia.

c) La persona que realiza la notificación de la incidencia.

d) A quién se le comunica la incidencia.

e) Los efectos derivados de la incidencia.

3.6. Política y Regulación.

R2 (SYRS36S26) Los usuarios autorizados del sistema de información deberán conocer sus obligaciones relacionadas con los controles de acceso y la información que tienen a su cargo. Para ello se establecen tres condiciones:

R1.1 Los usuarios autorizados tendrán que usar su clave de manera adecuada.

R1.2 Los usuarios autorizados no pueden descuidar ni un momento la información que manejan.

R1.3 Los usuarios autorizados tienen que seguir las medidas de seguridad impuestas para evitar accesos no autorizados a la información que manejan.

Trace-to: R1, R3 (...)

SRS *(Especificación de Requisitos del Software)*

3. Requisitos Específicos.

3.5. Atributos del Software.

3.5.3. Seguridad.

3.5.3.1. Confidencialidad.

R3 (SRS3531S1) El sistema operativo que se utilice proporcionará el mecanismo de claves para controlar y/o limitar el acceso a los usuarios.

Trace-to: R4

R4 (SRS3531S14) El sistema operativo que se utilice proporcionará programas para verificar la calidad de las contraseñas en el Sistema de Control de Accesos. Se dice que una clave es de calidad si cumple por lo menos estas tres características:

a) El número mínimo de caracteres es [n, n >= 6].

b) Tiene al menos [n, n>=1] caracteres numéricos y [m, m>=1] caracteres alfanuméricos.

c) Se cambia cada [tiempo en días] para usuarios generales y cada [tiempo en días] para usuarios con privilegios.

Figura 12 Ejemplos de requisitos de seguridad del sistema y del software

2.7.2 SIREN y la auditoría informática de seguridad

Una auditoría informática consiste en la realización de exámenes periódicos sobre un sistema de información con la finalidad de analizar y evaluar la planificación, el control, la eficacia, la seguridad, la economía y/o la adecuación de la infraestructura informática de la empresa. La auditoría de seguridad, en particular, consiste en la revisión y examen independientes de los registros y actividades de un sistema a fin de verificar si los controles del sistema son adecuados, para garantizar el cumplimiento con la política establecida y con los procedimientos operativos, para detectar problemas de seguridad, y para recomendar posibles cambios en la política de control y en los procedimientos [146].

Una auditoría de seguridad puede abarcar muchos aspectos, como puede ser el grado de protección de las instalaciones o de las personas. En los últimos años ha crecido el interés por garantizar un tratamiento adecuado de los datos de carácter personal. En el proyecto fin de carrera de Miguel Ángel Martínez Aguilar [205], cuyo objetivo principal era la mejora de los catálogos Seguridad-PDP y Seguridad-MAGERIT, se observó la posibilidad de utilizar el catálogo Seguridad-PDP como soporte en la auditoría de datos de carácter personal. Este doctorando colaboró en los primeros trabajos en esta línea [206, 207], que después ha seguido evolucionando independientemente [208, 209].

La idea original era sencilla: el uso de los requisitos predefinidos en el catálogo Seguridad-PDP en el marco de un proceso estándar de auditoría puede proporcionar a las organizaciones una garantía de que se asegura adecuadamente la privacidad y el buen uso de los datos personales gestionados. Para ello, dentro de las actividades propiamente dichas de auditoría, el auditor realizará comprobaciones en el sistema de la organización auditada para verificar el cumplimiento o incumpliendo de los requisitos contenidos en el catálogo, que se manejan como una lista de chequeo. Estas comprobaciones se realizarán con el apoyo del personal responsable de la organización, que facilitará en la medida de lo posible la tarea del auditor.

El proceso de auditoría propuesto en este trabajo ha sido puesto en práctica en un caso de estudio de un centro médico de la ciudad de Murcia, que obviamente tiene un nivel alto de protección en los datos personales que trata. Este experimento ha sido diseñado siguiendo el método de Investigación-Acción (Sección 1.3.2). La organización en cuestión ofrece desde el año 1988 todos los servicios para el cuidado de la salud, atendiendo consultas terapéuticas y de diagnóstico relacionadas con diferentes especialidades médicas, que dan respuesta a las demandas de más de 5.000 pacientes mensuales. En [206, 207] se puede encontrar una descripción detallada de este caso de estudio.

2.7.3 Casos de estudio del proyecto GARTIC

En el marco del proyecto GARTIC, descrito en la Sección 1.4.3.1, se han desarrollado cinco casos de estudio, uno por cada una de las pequeñas y medianas empresas de desarrollo de software participantes. Este doctorando ha estado involucrado como tutor en tres de ellos, correspondientes a las empresas *Bahía IT S.A.*, *Foro Digital S.L.* y *Adalid Management & OutSourcing S.L.*, a las que por motivos de confidencialidad en este apartado hacemos corresponder aleatoriamente las etiquetas E1, E2 y E3.

El proyecto GARTIC se inició con un curso de formación en IR, SIREN y la herramienta CARE comercial Requisite Pro, y continuó con la implantación en dos fases de SIREN: (1) en la primera, se hizo una identificación de dominios de aplicación candidatos de interés para las empresas participantes (verticales y horizontales). Cada empresa designó un analista responsable del desarrollo de los proyectos. Después de seleccionar un dominio, se implantó independientemente SIREN *para* reutilización en cada empresa, y se obtuvo un catálogo de requisitos para cada dominio seleccionado, en un proceso que como mínimo involucró tres iteraciones entre los analistas de las empresas y los tutores del GIS; (2) en la segunda fase, se implantó SIREN *con* reutilización, desarrollando un proyecto a partir de los documentos de requisitos especificados en la primera fase, es decir, instanciando el catálogo de requisitos, al menos una vez, haciendo uso de la nueva versión de SirenTool (Sección 2.9). En las siguientes tablas (Tabla 13, Tabla 14 y Tabla 15) se muestra un resumen de los proyectos piloto, centradas en la primera fase, para reutilización.

EMPRESA-TIC	E1
Catálogo	*CatWeb*, un catálogo de funcionalidades de aplicaciones web de comercio electrónico
Tipo	Catálogo de Dominio (vertical)
Propósito	Documentar un desarrollo ya realizado
Método/técnicas de análisis que proponen usar	E1 no propone usar en el análisis ningún método o técnica de ingeniería del software
Documentos desarrollados en el catálogo	SRS
Estado del catálogo	Tres iteraciones sobre el catálogo, que comprende los 8 módulos comunes de la aplicación.
Número de requisitos identificados	75 requisitos

Tabla 13 Resumen proyecto piloto GARTIC Empresa E1 (desarrollo *para*)

EMPRESA-TIC	E2
Catálogo	*Suite NEXO*, que incluye un *Gestor de contenidos* que soporta las necesidades comunes de gestión web de contenidos y documentos, con funcionalidades de tramitación electrónica y firma digital y prestaciones multicanal (por ejemplo, para la Televisión Digital Terrestre).
Tipo	Catálogo de Dominio (vertical)
Propósito	Documentar un desarrollo ya realizado
Método/técnicas de análisis que proponen usar	E2 propone usar Métrica 3 (versión estructurada)
Documentos desarrollados en el catálogo	SRS
Estado del catálogo	Tres iteraciones sobre el catálogo, acotado al Gestor de contenidos
Número de requisitos identificados	273 requisitos

Tabla 14 Resumen proyecto piloto GARTIC Empresa E2 (desarrollo *para*)

EMPRESA-TIC	E3
Catálogo	*Armorknown*, requisitos reutilizables de todos los módulos implicados en una aplicación TPV (*Terminal de Punto de Venta*)
Tipo	Catálogo de Dominio (vertical)
Propósito	Documentar un desarrollo ya realizado
Método/técnicas de análisis que proponen usar	E3 proponen usar simplemente diagramas de contexto
Documentos desarrollados en el catálogo	SRS
Estado del catálogo	Tres iteraciones sobre el catálogo.
Número de requisitos identificados	277 requisitos

Tabla 15 Resumen proyecto piloto GARTIC Empresa E3 (desarrollo *para*)

Siguiendo con la terminología de la Sección 1.3.1, en el proyecto GARTIC se han desarrollado verdaderos casos de estudio, pues desde el principio del proyecto se había establecido un conjunto de medidas con el objetivo de cuantificar la experiencia. ECAPRIS (*Evaluación de la CAlidad y la Productividad al implantar un método de Ingeniería de Software*), es un método de medición ágil desarrollado por Duque *et al.* [88], basado en GQM (*Goal Question Metric*) y CMMI, que tiene por objetivo evaluar la mejora en calidad y productividad que conlleva la implantación de nuevos métodos de ingeniería de software en pymes. ECAPRIS se ha aplicado en GARTIC en las dos fases de desarrollo para y con reutilización, con el objetivo de evaluar las variaciones en la calidad y la productividad del proceso de gestión de requisitos y del producto software al implantar SIREN para producir un catálogo de requisitos y al menos un documento de requisitos específicos. Los resultados que Duque *et al.* [88] han medido en los proyectos piloto de las empresas E1-E3 se muestran en la Tabla 16.

El bajo nivel de madurez en cuanto al proceso de IR de las empresas participantes ha determinado que el proyecto se haya centrado no sólo en transferir un método de IR basado en reutilización, sino también un proceso de IR básico. Dos de las tres empresas no tenían experiencia con métricas del software, de manera que el proyecto también les ha servido para iniciarse en este campo. Todas las empresas entregaron los sucesivos cuestionarios y rellenaron la información de uso de la herramienta fuera de plazo.

Al implantar SIREN con reutilización, existe un consenso en las empresas participantes en mencionar beneficios como los que destacamos a continuación: (1) la utilización del catálogo de requisitos facilita la estimación del esfuerzo de sucesivos proyectos, mejorando de esta manera las estimaciones de tiempo y recursos; (2) el incremento en el uso de estándares de requisitos y el mejor conocimiento del dominio de aplicación redunda en mejoras en la documentación, trazas de requisitos y comunicación con el cliente; (3) la reducción de las desviaciones y la disminución del número de inconsistencias entre requisitos; y (4) la mayor agilidad en la validación, que permite que los requisitos se ajusten más rápidamente a las expectativas de los clientes.

El proceso de implantación de SIREN en las empresas ha permitido refinar las características de SirenTool (Sección 2.9), que ha sido revisada completamente e implantada en E1-E3 en la fase de SIREN con reutilización, obteniéndose así una útil retroalimentación. Para una discusión más detallada de los resultados cuantitativos y cualitativos de este proyecto remitimos a Duque *et al.* [88].

RESULTADOS			EMPRESAS		
			E1	E2	E3
SIREN para reutilización					
M1	#de req. del catálogo		75	273	277
M2	Esfuerzo de creación del catálogo (H/h)		30	45	300
M3	Esfuerzo por req. (H/h)	=M2/M1	0.40	0.16	1.08
SIREN con reutilización					
M4	# de req. reutilizados sin modificación		13	150	120
M5	Esfuerzo en inserción de requisitos (H/h)		2	30	6
M6	# de req. reutilizados con modificación		6	20	10
M7	Esfuerzo en modificación (H/h)		1.5	5	3
M8	# de req. parametrizados sin modificación		4	0	5
M9	Esfuerzo de inserción de requisitos parametrizables (H/h)		0.5	0	2
M10	# de req. nuevos		0	200	245
M11	Esfuerzo en creación (H/h)	=M3*M10	0	50	176
M12	% del catálogo reutilizado	=(M4+M6+M8)/M1	31%	62%	49%
M13	% de req. reutilizados	=(M4+M6+M8)/ (M4+M6+M8+M10)	100%	46%	36%
M14	% de esfuerzo de reutilización req.	=(M5+M7+M9)/ (M5+M7+M9+M11)	100%	41%	6%
M15	% de requisitos reutilizados sin modificaciones	=(M4+M8)/ (M4+M6+M8+M10)	74%	41%	33%
M16	Esfuerzo no invertido (H/h)	=(M4+M6+M8)*M3	9.2	28.0	146.21

Tabla 16 Resultados de las empresas E1-E3 en GARTIC (H/h: *Horas/hombre*)

2.8 Ocho cuestiones clave en reutilización de requisitos

La experiencia obtenida en la definición y puesta en práctica del método SIREN –resumida en la Sección 2.7–, junto con la investigación realizada a lo largo de los años en el campo de la reutilización de requisitos y especialmente de las herramientas CARE, nos ha llevado a formular en el GIS ocho cuestiones clave que, en nuestra opinión, deberían ser soportadas de una u otra manera por cualquier método de IR basado en reutilización. Estas cuestiones clave se presentan en Toval *et al.* [280] y son las mostradas en la Tabla 17.

Las ocho cuestiones clave sintetizan una suerte de lecciones aprendidas en las experiencias mostradas en la Sección 2.7, y han servido (1) para reflexionar sobre las principales aportaciones y los aspectos más destacados de la investigación en IR y reutilización presentada en este capítulo; y (2) para identificar las principales cuestiones relacionadas con la reutilización que deberían ser soportadas por un método de IR

basado en reutilización y en particular por una herramienta CARE que soportara un método de IR basado en reutilización. Estas ocho cuestiones clave han sido fundamentales a la hora de establecer un marco para revisar el estado del mercado de herramientas CARE en relación con la reutilización, como se muestra en la Sección 2.9.1. Se trata de cuestiones sencillas: uno de los objetivos de la línea de investigación de IR y reutilización, como se afirma en la Sección 1.1, es acercar el estado del arte y el estado de la práctica. Con la propuesta de un método práctico y sencillo, y a través de la satisfacción de estas cuestiones clave, pensamos que una empresa con un nivel bajo de madurez puede dar un primer paso en la mejora de su proceso de IR.

K1. *Organización de los requisitos reutilizables*
K2. *Máquina de búsqueda de requisitos reutilizables*
K3. *Selección y reutilización de requisitos con distintos niveles de granularidad*
K4. *Reutilización de atributos de los requisitos*
K5. *Reutilización de relaciones de traza*
K6. *Gestión de requisitos parametrizados*
K7. *Mejora del repositorio*
K8. *Soporte CARE para la reutilización*

Tabla 17 Ocho cuestiones clave en reutilización de requisitos

Se trata de cuestiones clave que en todos los casos están íntimamente ligadas al soporte automatizado del método de IR que contempla reutilización (especialmente K2, K3 y K8). En Toval *et al.* [280] se muestra un estudio de tres de las principales herramientas CARE comerciales, que revela una carencia de soporte automatizado a estas ocho cuestiones, lo cual nos ha llevado a proponer una herramienta propia de soporte de SIREN, *SirenTool*, y a mostrar cómo esta herramienta soporta cada una de las cuestiones clave [280].

Estas ocho cuestiones clave y la viabilidad de la herramienta desarrollada han sido evaluadas en el contexto de un caso de estudio para el desarrollo de un sistema de información para la gestión de historias clínicas en la unidad de cuidados intensivos del Hospital de Getafe en Madrid, en colaboración con Manuel Campos, miembro del equipo de desarrollo de dicho proyecto de transferencia tecnológica [3].

2.8.1 Organización de los requisitos reutilizables (K1)

Compartimos la visión de Lam *et al.* [171] en el sentido de que la definición de requisitos reutilizables se beneficia de disponer de una guía para estructurar el dominio y para organizar los requisitos reutilizables. Para facilitar la reutilización es fundamental que los requisitos reutilizables estén bien organizados y estructurados, pues si es así el tiempo para extraer requisitos del repositorio se reduce.

En SIREN, por ejemplo, hemos elegido organizar los requisitos reutilizables por medio de catálogos que usan unas plantillas de documentos predefinidas. Un catálogo agrupa un conjunto de requisitos relacionados que se corresponde con un dominio de aplicación o con un perfil. Otras estructuras también sirven para ejemplificar K1: libros de patrones (*pattern book*) [264], casos de uso [23] o mapas de estructura (*structure maps*) [247].

2.8.2 Máquina de búsqueda de requisitos reutilizables (K2)

Un método de IR basado en reutilización debe tener disponible una máquina de búsqueda fácil de usar que satisfaga todas las necesidades de consulta de los ingenieros de requisitos en tiempo de reutilización. La capacidad de seleccionar los requisitos más adecuados para un nuevo proyecto es fundamental. El ingeniero de requisitos debe ser capaz de definir un amplio rango de criterios de búsqueda para realizar la selección de requisitos reutilizables. Esta cuestión clave está directamente ligada con la cuestión clave K8, "Soporte CARE para la reutilización", pues la máquina de búsqueda debería estar implementada en el contexto de una herramienta de gestión de requisitos en la cual se debe proporcionar también la capacidad de seleccionar requisitos a partir del conjunto resultado de la búsqueda.

Por ejemplo, en el caso de utilizar plantillas de documentos, como en SIREN, puede ser necesario buscar los requisitos especificados en una sección particular del documento o aquellos requisitos relacionados con una determinada incumbencia de los interesados (como por ejemplo, "cuestiones legales"). También puede ser necesaria la búsqueda de requisitos que contengan una palabra específica, como los requisitos que contengan la palabra "firewall" en el catálogo Seguridad-MAGERIT.

2.8.3 Selección y reutilización de requisitos con distintos niveles de granularidad (K3)

Se debe poder definir en tiempo de reutilización el tamaño de grano de lo reutilizado en el nivel de requisitos. El conjunto de requisitos reutilizados debería poder estar formado por un único requisito, por un subconjunto de requisitos procedentes de algunos de los catálogos disponibles, o por un catálogo completo.

Existen aproximaciones a la reutilización de requisitos en las que la granularidad es fija. Por ejemplo, aquellas basadas en un *documento base* de requisitos. Esta clase de reutilización es menos eficiente, pues es obligatorio reutilizar un conjunto determinado de requisitos, cuando sólo se requiere un subconjunto de los mismos. Si no se permiten distintos niveles de reutilización, la reutilización se hace menos eficiente pues se debe dedicar un tiempo a la eliminación de requisitos que no son necesarios para el nuevo proyecto.

2.8.4 Reutilización de atributos de los requisitos (K4)

Un requisito reutilizable debe estar descrito por medio de un conjunto de atributos. Estos atributos se deben manejar cuidadosamente en tiempo de reutilización, de manera que cuando un requisito es reutilizado sus atributos también lo sean. El ingeniero de requisitos debe tener disponible la flexibilidad de poder modificar cualquier atributo en tiempo de reutilización. Cuando el ingeniero de requisitos decide cambiar el tipo del requisito reutilizado puede aparecer un problema, porque el conjunto de atributos que pertenece a cada tipo de requisito, origen y destino, puede ser distinto. Por ejemplo, un requisito de protección de datos puede ser reutilizado como un requisito de seguridad en el caso de los catálogos de SIREN Seguridad-PDP y Seguridad-MAGERIT.

Para ayudar a mantener la consistencia en tiempo de reutilización es necesario definir un conjunto mínimo de atributos que todos los tipos de requisitos deben tener en común.

En SIREN se ha definido un conjunto mínimo de atributos, MIN, basado en el estándar IEEE 1233 [139] (Sección 2.6.2). Por otro lado, el valor de los atributos de los requisitos puede variar de un proyecto a otro pero la lista de atributos englobados por un tipo de requisito ha de permanecer en tiempo de reutilización. Posteriormente, a través del desarrollo del proyecto actual, se pueden definir nuevos atributos para ese tipo de requisito.

2.8.5 Reutilización de relaciones de traza (K5)

Normalmente los requisitos reutilizables no están aislados sino que están relacionados unos con otros por medio de trazas. Obviamente la existencia de estas trazas se debe tener en cuenta en tiempo de reutilización. En tiempo de reutilización el ingeniero de requisitos debe conocer las relaciones de traza entre requisitos de manera que pueda decidir qué requisitos deben permanecer en el nuevo proyecto. Es útil que el soporte automatizado permita que estas relaciones de traza se puedan visualizar gráficamente de forma adecuada, de manera que puedan ayudar en el proceso de toma de decisiones en tiempo de reutilización. Además, cuando la especificación de requisitos evoluciona, las relaciones de traza pueden ser utilizadas para evaluar el impacto del cambio de un requisito en los requisitos que dependen de él. La gestión de cambios se puede beneficiar así de una adecuada gestión de las trazas.

Ramesh y Jarke [256] proponen una amplia taxonomía de trazabilidad, pero (1) como afirman von Knethen *et al.* [293], no indican qué tipo de dependencias deben ser trazadas para soportar reutilización; y (2) en nuestra opinión se trata de una taxonomía demasiado compleja para ser utilizada en un proceso de IR práctico como el que se pretende definir con SIREN. Por lo tanto, proponemos un conjunto reducido de relaciones de traza *horizontales*, que son aquellas trazas que relacionan artefactos en el mismo nivel de abstracción, en este caso, en el nivel de los requisitos. Los requisitos relacionados pueden ubicarse en el mismo documento (por ejemplo, el SRS) o en distintos documentos (por ejemplo, consideremos un requisito del SYRS que se relaciona con un requisito del SRS). El modelo de traza propuesto para SIREN se presenta en la Sección 2.6.4. Este modelo de traza revisa el de Toval *et al.* [280] a través de la experiencia del proyecto GARTIC (Sección 1.4.3.1).

2.8.6 Gestión de requisitos parametrizados (K6)

La parametrización es un concepto que ha sido extensamente utilizado en programación para favorecer la reutilización. En el nivel de los requisitos, dejando al margen aproximaciones formales como por ejemplo la de OBJ3 [120] (un lenguaje basado en reescritura de términos), la parametrización ha sido trasladada en forma de (1) *características parametrizadas* (*parameterized features*) como las de FODA [160] y FORM [161]; y especialmente en (2) *requisitos parametrizados* (*parameterized requirements*) (ver por ejemplo [106, 171]). Los requisitos parametrizados contienen parámetros, es decir, partes que deben ser adaptadas a cada aplicación o sistema y a las que se debe dar valores en tiempo de reutilización o en tiempo de especificación.

Los requisitos parametrizados promueven la reutilización factorizando detalles específicos de la definición del sistema en forma de parámetros del requisito, de manera que se permite a los ingenieros de requisitos especificar requisitos con un nivel de flexibilidad mayor. Es interesante además que en tiempo de reutilización el ingeniero de

requisitos disponga de un conjunto de valores adecuados. El modelo de requisitos parametrizados propuesto para SIREN se ha presentado en la Sección 2.6.3. De nuevo este modelo revisa el de Toval *et al.* [280] a través de la experiencia del proyecto GARTIC (Sección 1.4.3.1 y Sección 2.7.3).

2.8.7 Mejora del repositorio (K7)

El repositorio no se debe considerar un producto estático sino evolutivo, dado que uno de los beneficios de la reutilización es la mejora continua de la calidad de los productos reutilizados. A través de la experiencia que se gana con su uso, el repositorio se mejora (1) para mejorar la calidad de los requisitos reutilizables en él incluidos, mejorando su adecuación, solventando ambigüedades, incompletitudes y errores en su especificación, añadiendo nuevas relaciones de traza, nuevos posibles valores para los parámetros, etc.; y (2) para extender el contenido con nuevos requisitos reutilizables. Para permitir esta evolución se debe definir un proceso de gestión de la configuración del repositorio, que implemente en detalle la actividad T8 de SIREN, "Mejora del repositorio".

2.8.8 Soporte CARE para la reutilización (K8)

El soporte CARE facilita la aplicación de la IR o, simplemente, lo hace posible [275]. Sin soporte automático, la aplicación de cualquier método de IR que contemple la reutilización de requisitos se transforma en lenta y engorrosa, y por tanto, impráctica. Varias de las cuestiones clave anteriores (a saber, K2, K3, K4, K5 y K6) sólo tienen verdadero sentido en un proceso que está respaldado por un soporte automático. Firesmith [104] considera la reutilización de requisitos una propiedad crítica en las herramientas de requisitos. Sin embargo, como se muestra en la Sección 2.9, esta propiedad no se soporta suficientemente en las principales herramientas CARE comerciales. Esta es la razón que nos ha llevado a desarrollar SirenTool.

2.9 Herramienta de soporte: SirenTool

La herramienta de soporte a SIREN se denomina SirenTool. En [175, 177, 280] se encuentran referencias a un prototipo inicial en cuya definición ha participado este doctorando y que ha sido desarrollado por el doctorando del GIS Joaquín Lasheras. Posteriormente, a lo largo del proyecto GARTIC Joaquín Lasheras ha revisado completamente este prototipo inicial y ha desarrollado una nueva versión de la herramienta, SirenTool 2.0.

2.9.1 La reutilización en las herramientas CARE comerciales

Lasheras *et al.* [177] y Toval *et al.* [280] muestran sendos estudios sobre cómo tres de las principales herramientas CARE comerciales soportan las características básicas de reutilización. En esta sección resumimos estos estudios. Las herramientas escogidas para la comparación han sido DOORS, Requisite Pro y Caliber-RM (ver Sección 2.2.5). Se han escogido estas tres herramientas por (1) ser herramientas comerciales bien conocidas (especialmente las dos primeras); (2) aparecer en todas las comparaciones de

herramientas que conocemos (mostradas en la mencionada Sección 2.2.5); (3) ser las únicas referenciadas en un interesante artículo de Kaindl *et al.* [158] sobre el estado de la IR; y (4) proporcionar un interfaz de extensibilidad que permite su adaptación.

En la comparación se han adoptado dos aspectos: (1) las ocho cuestiones clave descritas en la Sección 2.8; y (2) las necesidades generales de una herramienta de gestión de requisitos. La Tabla 18 resume las características consideradas necesarias para nuestra herramienta de gestión de requisitos. La tabla sólo muestra las cuestiones clave para la reutilización pues las tres herramientas satisfacen las características esenciales de una herramienta de gestión de requisitos, como muestra el *survey* de INCOSE sobre herramientas CARE [144]. La cuestión K8, "Soporte CARE para la reutilización", no se muestra en la tabla pues su valoración se deriva de las siete anteriores.

Cuestión Clave	***REQUISITE PRO***	***CALIBER-RM***	***DOORS***
K1. Organización de los requisitos reutilizables	Sí, por tipo de requisito		Sí, por tipo de objeto
K2. Máquina de búsqueda de requisitos reutilizables	Los requisitos no pueden ser seleccionados a partir del resultado de las búsquedas		
K3. Selección y reutilización de requisitos con distintos niveles de granularidad	Copiar y pegar y documentos base		Copiar y pegar y módulos
K4. Reutilización de atributos de los requisitos	Cuando el requisito se copia y pega		
K5. Reutilización de relaciones de traza	Sólo trazas padre-hijo	No	
K5.1. Requisitos exclusivos	No		
K5.2. Muestra relaciones de traza existentes	Sí, gráficamente		
K5.3 Gestión de inconsistencias	Sí, gráficamente con una matriz de trazas		
K6. Gestión de requisitos parametrizados	No		
K7. Mejora del repositorio	No se trata explícitamente		

Tabla 18 Resumen del soporte prestado a las cuestiones clave por parte de las herramientas CARE seleccionadas

Este estudio concluye que ninguna de las tres herramientas proporciona adecuadamente reutilización de requisitos pues ninguna de ellas soporta todas las cuestione clave para la reutilización de requisitos. La única forma de reutilizar es mediante copia y pega de requisitos ya definidos. Ninguna de las herramientas ofrece gestión de requisitos parametrizados, reutilización de relaciones de traza, o gestión de requisitos exclusivos. Matulevicius [211] también incide en las limitaciones de las herramientas CARE para soportar reutilización de requisitos.

2.9.2 SirenTool

A partir de las ocho cuestiones clave para reutilización de requisitos y de los resultados del estudio anterior sobre el soporte de reutilización por parte de las principales herramientas CARE comerciales se ha desarrollado una especificación de requisitos del software de una herramienta CARE de soporte al método SIREN [175]. A la luz de esta especificación de requisitos ninguna de las herramientas analizadas en el apartado

anterior era adecuada para ser adoptada directamente para soportar reutilización de requisitos, de manera que se consideraron dos opciones para proporcionar un soporte automatizado a SIREN: (1) adaptar una de estas herramientas; o (2) desarrollar una nueva herramienta desde el principio. Se ha preferido la primera opción, pensando que es más efectiva en coste al tiempo que permite satisfacer completamente las necesidades de SIREN. Se ha escogido Requisite Pro como herramienta base para resolver los problemas identificados en la Tabla 18, debido fundamentalmente a la experiencia previa del grupo en el uso de esta herramienta [284, 285].

SirenTool 2.0 consta de la funcionalidad completa de la herramienta Requisite Pro más cuatro nuevas aplicaciones que incluyen funcionalidades relacionadas con la reutilización. Estas cuatro nuevas aplicaciones, a las que llamaremos conjuntamente, *SIRENA* (*SIREN Automated*), son las siguientes: (1) *SIREN CON REUTILIZACIÓN*; (2) *SIREN PARA REUTILIZACIÓN*; (3) *INSTANCIAR PARAMETRO*; y (4) *ESTADISTICAS*. Una descripción completa de SirenTool 2.0 queda fuera del ámbito de esta tesis doctoral. Con fines meramente ilustrativos, en este apartado nos limitamos a mostrar una de las principales ventanas de la aplicación, la interfaz de reutilización de requisitos, perteneciente a SIREN CON REUTILIZACIÓN (Figura 13). En Nicolás *et al.* [226] se puede encontrar una guía completa sobre todas las capacidades de esta herramienta.

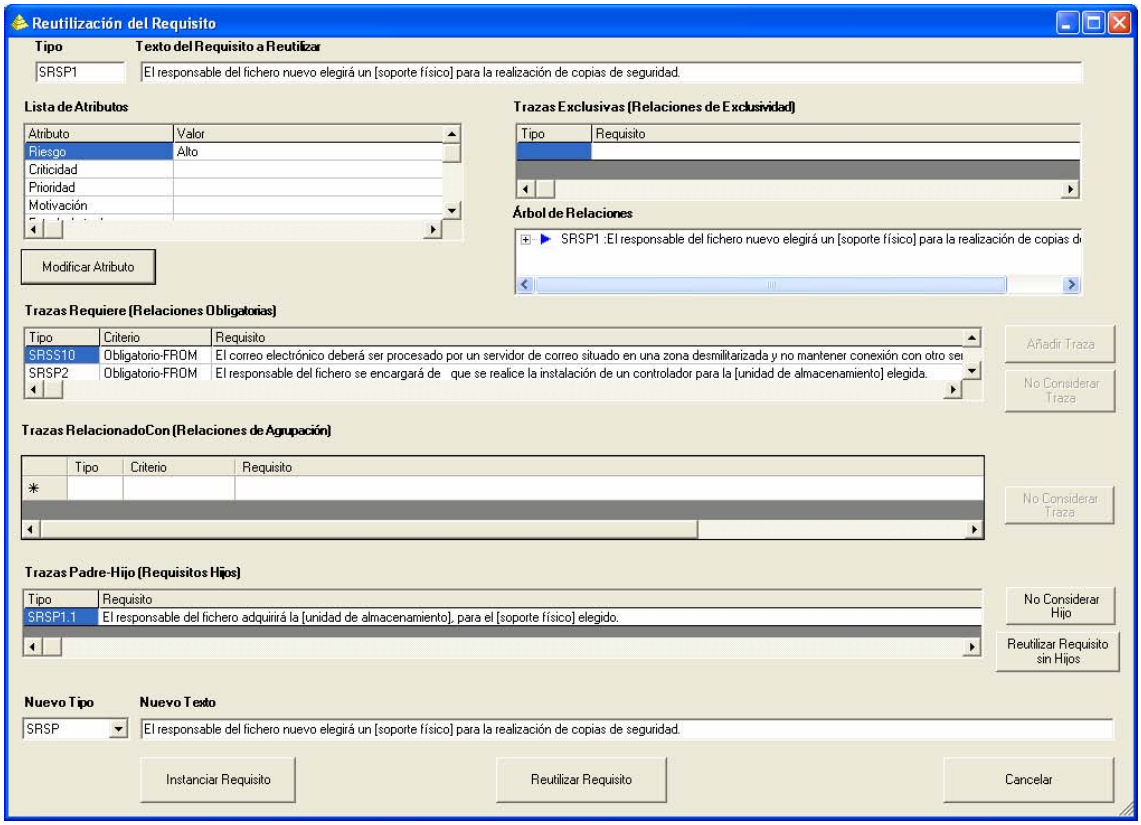

Figura 13 Interfaz para la reutilización del requisito de SirenTool 2.0

2.10 SIRENgsd: SIREN para el desarrollo global de software

2.10.1 Introducción

En los últimos años, el proceso de progresiva globalización que está sufriendo la industria en general ha alcanzado la ingeniería del software. Con el cambio de siglo, y cada vez más frecuentemente, se publican documentos en los que se estudian diversos aspectos de la ingeniería del software en un ambiente global, que ha pasado a ser una realidad. Este nuevo modelo de desarrollo de software, en el que los participantes se encuentran distribuidos, a veces en varios países e incluso en continentes distintos, se conoce como *Desarrollo Global de Software* (DGS, en inglés *Global Software Development*) o *Desarrollo Distribuido de Software* (*Distributed Software Development*). Esta nueva forma de desarrollo de software permite (1) el aprovechamiento de los bajos costes de mano de obra en ciertos países que sin embargo cuentan con profesionales cualificados; (2) el acceso al conocimiento experto allí donde se encuentra; (3) la consecución de jornadas laborales de 24 horas aprovechando la diferencia horaria (modelo de trabajo *around-the-clock*); (4) el acceso global a un conjunto de recursos; y (5) la mejora de la proximidad al cliente o a ciertos mercados. Con este nuevo modelo de trabajo muchas de las actividades comprendidas dentro de la IR tienen que cambiar, al no poder ser llevadas a cabo como si los participantes implicados estuvieran colocalizados, tal como ocurre en el modelo tradicional de desarrollo de software, no distribuido.

Cuando Cheng y Atlee [70] estudian el estado de la investigación en IR, identifican la globalización como uno de los *puntos calientes* (*hotspots*) que merecen atención en la futura investigación en IR. La superación de las dificultades que el DGS plantea a la IR es fundamental para conseguir un eficiente desarrollo distribuido de productos software. Los requisitos deberían ser analizados, verificados, reutilizados, trazados, gestionados y evaluados con respecto a los riesgos y a su cumplimiento a lo largo del proceso de desarrollo [151]. El principal riesgo inherente al GSD son los problemas de comunicación motivados por la distribución del equipo de desarrollo [70]. Por ejemplo, en una actividad intrínsecamente colaborativa como la extracción y el modelado inicial (*early modeling*) de requisitos, se producen tensiones contrapuestas entre la distancia que impone el desarrollo global y la necesidad de construir un modelo mental de las necesidades y requisitos del proyecto que sea compartido por todos los participantes en el mismo. La distancia no se debe medir sólo en términos geográficos, sino también en términos de husos horarios, lengua y cultura. Para Cheng y Atlee dos son los desafíos más importantes en la IR para DGS: (1) por un lado, se necesita idear nuevas técnicas de IR, o extender las actuales, de forma que se soporte la *externalización* (*outsourcing*) de las actividades de desarrollo posteriores a la IR, como el diseño, la codificación y la prueba; (2) por otro lado, se necesitan técnicas de soporte para facilitar y gestionar la extracción de requisitos, el modelado, la negociación de requisitos y la gestión de equipos en el proceso de IR distribuida.

Siendo la Ingeniería del Software una disciplina joven, el DGS lo es mucho más. Aún tratándose de una problemática relativamente poco estudiada, en los últimos años se ha publicado un cierto número de estudios que identifican los problemas que surgen al intentar desarrollar las actividades de IR en un ambiente global, y que proponen soluciones a tales problemas: superar las dificultades que el DGS plantea a la IR será fundamental para conseguir un desarrollo distribuido y eficiente de software.

Tanto el DGS como la reutilización de requisitos son dos aproximaciones que prometen beneficios para la industria. En la medida de nuestro conocimiento, no existe ninguna propuesta para dar solución a estas dos cuestiones de forma conjunta. Sin embargo, en la literatura se reporta con frecuencia que uno de los problemas del DGS es que la gestión del conocimiento no es óptima, pues existe gran cantidad de conocimiento generado en sitios remotos que no se comparte adecuadamente con los desarrolladores. La gestión del conocimiento de forma distribuida es problemática, como también lo es la reunión de los requisitos extraídos en distintas localizaciones. Siguiendo la línea de investigación de IR y requisitos de esta tesis doctoral, todo esto parece sugerir la necesidad de que en el DGS exista algún mecanismo que permita compartir conocimiento, y en particular requisitos, por ejemplo a través de algún tipo de repositorio. Además, en este contexto de desarrollo distribuido de software suelen aparecer múltiples vistas e interpretaciones tanto de los requisitos como del resto de modelos que forman parte del desarrollo, por lo que se hace imprescindible la gestión extensible de la consistencia entre las distintas vistas y modelos que formarán parte del desarrollo global.

En este apartado se presenta *SIRENgsd* (donde *gsd* procede de *global software development*), una extensión de SIREN que se ha diseñado con el fin de obtener un método basado en reutilización para la IR global (IRG). Esta extensión no figuraba entre los objetivos iniciales de la tesis (Sección 1.2), pero el interés que el DGS ha suscitado recientemente y la ausencia de un método de IRG que aborde expresamente la reutilización nos ha llevado a considerar este nuevo campo de conocimiento en la línea de investigación de IR y reutilización. Para ello en primer lugar se ha estudiado el estado del arte, atendiendo a los problemas identificados en la literatura en relación con el DGS, y examinando las soluciones propuestas (Sección 2.10.2). Con base en este estudio se ha propuesto un catálogo de buenas prácticas para gestionar las amenazas que afectan la IRG que han sido identificadas en la revisión sistemática (Sección 2.10.3). Con base en este catálogo se propone un método de IRG como una extensión de SIREN que materializa las guías del catálogo de buenas prácticas: se propone un modelo de procesos, técnicas, y herramientas (Sección 2.10.4).

2.10.2 Riesgos y salvaguardas en la IR global: revisión sistemática

Para estudiar el estado del arte se ha realizado una Revisión Sistemática de la Literatura (RSL, ver Sección 1.3.3) [163]. La presentación de esta Sección 2.10.2 está inspirada en la de Kitchenham *et al.* [164].

2.10.2.1 Ámbito de la revisión

En primer lugar definimos dos conceptos que sirven como base a la RSL: (1) *riesgo* o *amenaza*, una cuestión de la aplicación de la IR en el DGS que puede poner en peligro

el proceso de IR si se menosprecia y no te tiene en cuenta; los *desafíos* de la IR en el DGS se recogen también en este grupo de riesgos; (2) *salvaguarda*, una buena práctica, una recomendación o consejo que ayuda a resolver o controlar el/los riesgo/s correspondiente/s. Una salvaguarda no es siempre la solución completa ante un riesgo determinado, pero ayuda a controlarlo.

A partir de estas dos definiciones, con esta revisión sistemática se persiguen dos objetivos esenciales: (1) encontrar los riesgos que se han identificado en la literatura en la aplicación de la IR en los proyectos de DGS, y que no son comunes al proceso de desarrollo tradicional, no distribuido; y (2) encontrar las salvaguardas propuestas en la literatura para resolver o controlar tales riesgos.

2.10.2.2 Cuestiones de la investigación

A partir de los objetivos anteriores se han formulado dos preguntas:

RQ1. *¿Qué riesgos se han identificado en la literatura en la aplicación de la IR en el DGS?*

RQ2. *¿Qué salvaguardas han sido propuestas en la literatura para resolver o controlar los riesgos identificados en RQ1?*

2.10.2.3 Proceso de búsqueda

Para buscar información en relación con esta RSL se han seleccionado las siguientes fuentes:

- IEEE Digital Library (www.computer.org/portal/site/csdl/index.jsp)
- ACM Digital Library (portal.acm.org)
- Science@Direct (www.sciencedirect.com)
- MetaPress (Kluwer+Springer) (www.metapress.com)
- Wiley InterScience (www.interscience.wiley.com).
- Google Scholar (scholar.google.com)

La cadena de búsqueda utilizada en estas fuentes para responder conjuntamente a RQ1 y a RQ2 ha sido la siguiente: “Global Software Development” AND “Requirements Engineering”.

2.10.2.4 Criterios de inclusión y exclusión

Los criterios de inclusión que han sido aplicados para marcar los estudios encontrados como candidatos a ser seleccionados son los siguientes: se incluyen solamente los estudios que tratan los riesgos que han sido identificados en la investigación de la IR en el DGS, o que identifican soluciones a tales riesgos.

Los criterios de exclusión, que sirven para filtrar los estudios candidatos como finalmente seleccionados o no, son los siguientes: (1) los estudios seleccionados deben referirse principalmente al GSD, no simplemente mencionarlo de manera tangencial, y

no se deben referir al GSD desde el punto de vista de la educación, por ejemplo presentando métodos de enseñanza global; (2) se excluyen estudios publicados antes del año 2000; (3) se excluyen estudios con una extensión menor de cuatro páginas; y (4) obviamente aquellos estudios idénticos que aparecen duplicados en distintas fuentes de búsqueda son excluidos; se ha optado por asignar el estudio a la primera fuente en la que ha aparecido.

2.10.2.5 Evaluación de la calidad

En esta RSL no se han determinado mecanismos específicos para medir la calidad de las contribuciones. Con los criterios de exclusión (2) y (3) anteriores se ha tratado de mejorar la calidad de los estudios seleccionados. Además todas las fuentes de información, salvo Google Scholar, proporcionan estudios académicos publicados en revistas y congresos de calidad. Google Scholar también busca trabajos pertenecientes a foros académicos, pero además puede ser fuente de literatura gris, que en un campo tan reciente como este, puede tener un interés especial: informes técnicos, *white papers*, proyectos fin de carrera, etc.

2.10.2.6 Recogida de datos

Para cada estudio seleccionado se ha rellenado un formulario de extracción de datos adaptado de Biolchini *et al.* [42] (ver Tabla 19). Este formulario consiste de (1) una sección de resultados objetivos que corresponden a lo escrito por los autores del estudio; y (2) una sección de resultados subjetivos que relaciona las impresiones de los revisores con respecto al tema de esta RSL y las extrapolaciones que se pueden realizar. La sección de resultados objetivos recoge (1) el método de investigación del estudio; (2) los problemas y limitaciones reportadas por los autores (si las hay); (3) los problemas encontrados; (4) las soluciones propuestas; y (5) problemas y limitaciones del estudio reconocidos por los propios autores.

Debido a la naturaleza esencialmente narrativa de los datos recogidos éstos no han sido tabulados durante el análisis de datos.

2.10.2.7 Resultados de las búsquedas

Al aplicar las cadenas de búsqueda se ha encontrado un conjunto inicial de 564 estudios en las fuentes de información. Después, al aplicar los criterios de inclusión sobre este conjunto inicial se ha definido un conjunto de 175 estudios candidatos. Seguidamente se han aplicado los criterios de exclusión y se ha obtenido un conjunto final de 36 estudios seleccionados. Durante el proceso de selección, se ha leído el resumen y, cuando ha sido necesario para tomar una decisión, partes seleccionadas de los estudios. La Tabla 20 muestra, dividido por fuentes, el conjunto de estudios encontrados, candidatos y finalmente seleccionados. En la tabla no se muestran las fuentes de información en las cuales no se han seleccionado ningún estudio. El protocolo de la revisión ha sido ejecutado por Alejandro López Lorca como parte de su proyecto fin de carrera [185], y este doctorando ha revisado el protocolo, los artículos incluidos y excluidos, y ha discutido los resultados con él.

2.10.2.8 Síntesis de resultados

Después de examinar los estudios seleccionados, para clasificar estos trabajos se ha utilizado la siguiente taxonomía de seis categorías:

- *Estudios que exclusivamente identifican riesgos* en la aplicación de la IR al DGS, en los cuales no se propone ninguna solución.
- *Estudios que realizan propuestas "teóricas" exclusivamente*, que no se han validado: son propuestas que no tienen una aplicación directa, y no se proporcionan guías para ponerlos en práctica inmediatamente.
- *Estudios que proponen métodos*, para superar ciertos riesgos.
- *Estudios que proponen herramientas*, que ayudan a aliviar ciertos riesgos.
- *Estudios que ofrecen buenas prácticas* para hacer frente con éxito al DGS.
- *Miscelánea*: estudios que no caen en ninguna de las categorías anteriores.

Identificación del estudio	
Nombre	*El nombre del estudio, entre comillas.*
Autores	*Nombrados tal y como aparecen en el estudio.*
Revista o congreso	*Se indica la fuente del estudio.*
Procedencia	*Se especifica si el estudio procede de las búsquedas de la revisión sistemática, del conjunto de estudios disponibles antes de la revisión sistemática, de un trabajo relacionado de algún estudio revisado o de un documento presentado en algún congreso encontrado manualmente.*
Resultados objetivos	
Método del estudio	*El método que se ha utilizado para llegar a los resultados, si se ha reportado.*
Problemas encontrados	*Los problemas o desafíos que los autores han identificado en la aplicación de la IR al desarrollo global de software*
Soluciones propuestas	*Las soluciones que los autores proponen a los problemas hallados en la aplicación de la IR al desarrollo global de software.*
Problemas y limitaciones del estudio	*Aquellos expresamente identificados por los autores.*
Resultados subjetivos	
Impresiones generales / Posibles extrapolaciones	*Conclusiones personales acerca del trabajo; posibles conexiones con las líneas de trabajo actual.*

Tabla 19 Plantilla utilizada para la revisión sistemática

Fuente	**Estudios encontrados**	**Estudios candidatos**	**Estudios seleccionados**
IEEE	100	61	6
ACM	83	9	7
MetaPress	54	29	1
Google Scholar	327	76	22
Total	564	175	36

Tabla 20 Resultados de la revisión sistemática

2.10.2.9 Discusión

En López y Nicolás [185] se puede encontrar una discusión completa sobre los resultados de la RSL, incluyendo una descripción de cada uno de los trabajos seleccionados y su asignación a una de las seis categorías anteriores. En este apartado

nos limitamos a presentar un breve resumen de los aspectos que consideramos más relevantes de los estudios seleccionados, respondiendo conjuntamente a RQ1 y a RQ2.

A pesar de lo reciente de este tema, en los últimos años se han publicado trabajos que estudian los problemas que surgen al desarrollar las actividades de IR en un ambiente global. Cabe destacar en este punto el trabajo intensivo del grupo SEGAL de la Universidad Victoria de Canadá (http://segal.uvic.ca/), liderado por Damian, que ha estudiado numerosos aspectos de la IR en el ámbito del GSD, incluyendo los siguientes: (1) los problemas derivados de la distancia geográfica entre los participantes del proyecto [74]; (2) los derivados de las diferencias en dos dimensiones del trasfondo cultural de los interesados (la cultura étnica o nacional y la cultura corporativa o de la organización), la adquisición y compartición del conocimiento, las diferencias en los procesos aplicados y en las herramientas, y la comunicación y coordinación en el proyecto [77]; (3) la necesidad de mantener adecuadamente el *conocimiento global* (*awareness*) del proyecto [78, 166]. Este conocimiento global se refiere al conocimiento de las actividades del resto de los miembros de un grupo distribuido, y es especialmente importante cuando las contribuciones individuales son relevantes para la actividad grupal en sí; y (4) la negociación de requisitos, que adquiere una problemática especial cuando los interesados se hallan distribuidos y algunos de ellos deben tomar parte en la negociación de forma remota a través del ordenador [73, 75, 76, 79]. Además la colaboración de Damian con Zowghi ha dado por resultado varios estudios de los desafíos de la IRG y un conjunto de recomendaciones [80, 303, 304].

Illes-Seifert *et al.* [143] realizan un estudio empírico de los desafíos del desarrollo distribuido de software en general y de la IR distribuida en particular. Los problemas particulares de la IR distribuida se resumen en los siguientes: (1) la mitad de las barreras culturales que dan lugar a especificaciones ambiguas son barreras lingüísticas; (2) las barreras específicas del dominio implican distinta terminología o notación de requisitos; (3) las comunicaciones cara a cara no pueden ser sustituidas por comunicaciones indirectas; (4) las zonas horarias dificultan la comunicación; (5) se da una importante asincronía en la comunicación; (6) las responsabilidades no están definidas; (7) un alto número de interesados en el proyecto son fuente de requisitos; (8) en ocasiones los procesos tradicionales no son válidos para el nuevo entorno de desarrollo; y (9) en general no existe suficiente comunicación. Las propuestas de Illes-Seifert *et al.* frente a estos problemas incluyen las siguientes: (1) fomentar la comunicación cara a cara; (2) establecer comunicaciones siguiendo reglas formales, ciñéndose a una terminología común y utilizando herramientas; (3) asegurar la calidad mediante revisiones e inspecciones; (4) evitar confusiones culturales mediante entrenamiento (preparación y talleres); (5) proporcionar requisitos de ejemplo y un glosario de términos; (6) la estandarización de formatos; (7) definición de estándares mínimos para documentos; y (8) el uso de herramientas como videoconferencia, voz sobre IP y wikis. Decker *et al.* [84] afirman que la participación intensiva de los interesados incrementa la calidad de los requisitos extraídos y proponen el uso de wikis como plataforma para la elaboración distribuida asíncrona de los documentos de requisitos. Calefato y Lanubile [51] también proponen el uso de una herramienta que facilite las discusiones entre participantes globalmente distribuidos.

Bhat *et al.* [41] inspeccionan los problemas en las relaciones entre vendedores y clientes en torno a la IR, que se resumen en los siguientes: (1) conflicto entre los objetivos de cliente y vendedor; (2) baja implicación del cliente; (3) aproximaciones conflictivas al

proceso de IR; (4) desajuste del compromiso del cliente con los objetivos del proyecto; (5) desacuerdos en la herramienta seleccionada, (6) problemas de comunicación; (7) repudio de responsabilidad (la no disponibilidad del cliente causa que el proyecto sufra retrasos, que no son aceptados alegando que no es su responsabilidad); (8) problemas de autorización (el interesado proporcionado por el cliente no comprende suficientemente el dominio de aplicación); y (9) desajuste de las herramientas con las expectativas. Estos autores identifican factores estratégicos esenciales para el éxito de la relación cliente-vendedor en la IR distribuida: (1) objetivo compartido; (2) cultura compartida; (3) proceso compartido; (4) responsabilidad compartida; y (5) confianza. Una vez identificados los factores, proponen un marco de trabajo para seleccionar las mejores prácticas para satisfacer esos factores de éxito.

Prikladinicki *et al.* [252] identifican esencialmente desafíos de carácter cultural o comunicativo a través del estudio de una compañía que desarrolla software global. En este ámbito MacGregor *et al.* [189] identifican cinco patrones y anti-patrones culturales en el desarrollo global de software: (1) Patrón “Sí pero no”: una persona A solicita a una persona B que lleve a cabo una tarea, quien le contesta con un sí. Cuando llega la fecha límite, la tarea no se ha cumplido: la persona B no quería decir que sí haría la tarea, sino que sí le había oído/entendido. Se debe a diferencias culturales, por lo que habría que confirmar o volver a preguntar; (2) Patrón “Proxy”: hay personas que han vivido el tiempo suficiente en dos culturas como para hacer de intérpretes entre representantes de ambas ya que conocen los distintos valores éticos y las costumbres. Estos individuos deberían ser colocados en posiciones en las que puedan ejercer de intermediarios; (3) Anti-Patrón “Al pie de la letra”: los miembros de un equipo de trabajo siguen un método usado por otro equipo de trabajo aún sin entenderlo o considerarlo útil, sólo por mantenerlos contentos, con lo que el tiempo de desarrollo se prolonga al no comprender bien lo que están haciendo; (4) Anti-Patrón “Somos un único equipo”: en distintas culturas la concepción de equipo varía. En algunas se alienta el eliminar jerarquías, de modo que la comunicación puede fluir libremente entre el grupo, y se siente que las decisiones sobre un artefacto deben ser tomadas por aquellos que trabajan en él. En otras, se valora mucho la jerarquía y trato respetuoso, y se piensa que todas las decisiones deben ser tomadas por la unidad gestora central. Entre estos grupos puede haber problemas por tener unas concepciones tan distintas; y (5) Anti-Patrón “El cliente es el rey”: en algunas culturas no está bien visto que otros que no sean la dirección hablen con los clientes. Así la comunicación es poco fluida, y se molestará al cliente sólo cuando la dirección lo considere necesario. Es posible que algunos de los deseos del cliente no puedan llevarse a cabo, y no se le notifique. A partir de su experiencia en proyectos distribuidos en Siemens, Berenbach [40], en cambio, expone que los fallos debidos a problemas con la estructura organizacional o con la dirección eran mayores y más numerosos que los debidos a problemas con la extracción de requisitos o a malentendidos culturales.

Heindl *et al.* detectan carencias en la IR para el GSD en cuanto a trazabilidad [132] y herramientas CARE [133]: (1) la trazabilidad debe reflejar la motivación *(rationale)* de los requisitos y la traza entre requisitos y artefactos ulteriores del proceso software, como código o casos de prueba; (2) el soporte CARE debe permitir el intercambio efectivo y a tiempo de información (por ejemplo, en cuanto a la notificación de cambios en requisitos, tan pronto como se produzcan y únicamente a los interesados a los cuales resultan relevantes). La integración entre las herramientas usadas en el desarrollo y la

trazabilidad de requisitos a través de las fronteras de las herramientas son cuestiones que también deben ser tenidas en cuenta.

Damian [74] y Crofts *et al.* [62] afirman respecto a la gestión del conocimiento en entornos distribuidos que buena parte del conocimiento generado en sitios remotos no se comparte adecuadamente entre los desarrolladores. Gumm [128] estudia los problemas de la reunión de los requisitos extraídos en distintas *interfaces organizacionales*, que se definen como la relación entre dos unidades organizacionales que interactúan en un proyecto. Los problemas identificados por Gumm incluyen (1) la redistribución de los requisitos para su desarrollo con independencia de las interfaces de las cuales fueron extraídos; (2) el ajuste de requisitos en caso de que el producto en desarrollo deba ser embebido en otro ya completado o durante la negociación de los requisitos; y (3) la propagación de los cambios en los requisitos entre todas las interfaces organizacionales. Las distintas interfaces deberían, más que compartir un mismo proceso, conocer los procesos que usan las demás y sincronizarse. Estos trabajos reafirman la necesidad de mecanismos para compartir conocimiento, por ejemplo en forma de requisitos. Pensamos que se trata de problemas que podrían ser gestionados mediante la implantación de un repositorio global de requisitos reutilizables.

2.10.3 Un catálogo de riesgos y salvaguardas para la IR global

Uno de los resultados principales de la RSL ha sido la compilación de un catálogo de buenas prácticas en IR para el DGS [187], estructurado alrededor de los riesgos identificados, como una relación de riesgos y salvaguardas asociadas. La versión completa del catálogo consta de 106 riesgos y 52 salvaguardas y está disponible en línea [186]. Analizando los resultados de la RSL, los riesgos y salvaguardas se han ordenado en una taxonomía formada por siete categorías. En algunos casos los límites entre las distintas categorías están borrosos, en ocasiones las categorías parecen solaparse. Para cada riesgo se ha elegido la categoría que subjetivamente se ha considerado "predominante". Las categorías son las siguientes:

- *Comunicación y distancia*: riesgos derivados de la dispersión de los interesados entre países y zonas horarias distintas.
- *Dirección y coordinación del proyecto*: riegos derivados de las dificultades para el establecimiento de estructuras organizativas, definición de roles, y procedimientos de coordinación.
- *Clientes*: riesgos derivados de la interacción con clientes distribuidos.
- *Gestión del conocimiento global del proyecto (awareness)*: riesgos derivados de la dificultad de mantener el conocimiento global del proyecto *(awareness)* cohesivo y coherente cuando diferentes grupos de trabajo tratan de acceder a este conocimiento concurrentemente.
- *Diferencias culturales*: riesgos derivados de la interacción entre grupos que tienen rasgos culturales muy diferentes.
- *Herramientas que soportan los procesos*: riesgos derivados de la ausencia de herramientas adecuadas que soporten el proceso de IR de forma completa.
- *Miscelánea*: riesgos no ligados a ninguna de las categorías anteriores.

Los riesgos y salvaguardas se estructuran jerárquicamente: los más generales se van refinando en otros más específicos. Los riesgos y las salvaguardas tienen un atributo denominado *fuente*, que identifica el estudio seleccionado en la RSL a partir del cual fue extraído. Si se trata de una propuesta propia que no procede de ningún estudio, se etiqueta como *propuesta*. Cada salvaguarda tiene un atributo adicional, *controla* (*faces*), que se refiere al riesgo o riesgos que maneja, y que sirve para establecer trazas entre riesgos y salvaguardas (mostradas en las correspondientes matrices de traza amenazas/guías, una por cada una de las siete categorías anteriores [186]). Algunos riesgos tienen también un atributo *ejemplo* que sirve para ilustrar el problema evocado por el riesgo.

En la Tabla 21 se cuantifica cada una de las categorías del catálogo, mostrando los riesgos y las salvaguardas que conforman la categoría, el número de riesgos todavía no manejados por ninguna salvaguarda, y el número de salvaguardas no recogidas de la literatura sino propuestas por los autores. Un análisis pormenorizado del catálogo y en particular de estos datos se encuentra en López *et al.* [187].

En la Tabla 22 se muestran ejemplos de distintos tipos de riesgos y salvaguardas. Los roles implicados en SIRENgsd, algunos de los cuales se mencionan en estos ejemplos, se describen en la Sección 2.10.4.1.

	Riesgos		**Salvaguardas**	
Área de interés	**Número total**	**No cubiertos**	**Número total**	**Propuestas**
Comunicación y distancia	28	3	7	2
Dirección y coordinación del proyecto	22	0	11	7
Clientes	21	3	3	3
Gestión del conocimiento	10	2	4	4
Diferencias culturales	9	1	10	6
Herramientas	8	0	16	3
Miscelánea	8	7	1	0

Tabla 21 Cuantificación de los riesgos y salvaguardas del catálogo, por categorías

2.10.4 SIRENgsd: SIREN para el desarrollo global de software

En este apartado se hace una propuesta de extensión de SIREN para ser utilizado en ambientes de DGS. Para ello, en la Sección 2.10.4.1 se amplía el modelo de procesos de SIREN con nuevas tareas y se explica en detalle cada una de ellas. A continuación, en la Sección 2.10.4.2 se extiende el conjunto mínimo de atributos de los requisitos de SIREN y las relaciones de traza entre requisitos y se tratan brevemente las plantillas de documentos. Por último en la Sección 2.10.4.3 se describe de forma somera la herramienta de la que habría que disponer para dar soporte completo al proceso propuesto, esbozando sus características más relevantes.

Diferencias culturales

Riesgo: R19.3 En el DGS pueden aparecer barreras éticas en el trabajo, del tipo de que para algunos grupos es difícil expresar su insatisfacción a los clientes, y cuestiones clave y cuestiones accesorias (como ciertas cuestiones de apariencia, *look-and-feel*) pueden ser tratadas de la misma manera sin tener en cuenta las prioridades.
_fuente: Illes-Seifert *et al.* [143]

Salvaguarda: S12.1 Las dimensiones nacionales y culturales de Hofstede deberán ser estudiadas para comparar las distintas culturas.
_controla: R19, R19.2, R19.3, R19.4
_fuente: Aranda *et al.* [28]

Dirección y coordinación del proyecto

Riesgo: R25.2 En el DGS la carencia de un rol de analista centralizado, cualificado y firme puede afectar seriamente el desarrollo del proyecto.
_fuente: Berenbach [40]

Salvaguarda: S22 Se deberá establecer una jerarquía de liderazgo ágil y bien definida, con un analista jugando el rol de coordinador. Las responsabilidades del analista deberán estar bien definidas y éste deberá jugar su papel de forma cualificada y firme. En la jerarquía, por debajo del analista deberá haber representantes de equipo, que serán responsables de los analistas que pertenezcan a su grupo de trabajo.
_controla: R25, R25.1, R25.2, R26, R29, R29.1, R30, R33, R35
_fuente: Berenbach [40]

Herramientas que soportan los procesos

Riesgo: R38.3 En el DGS existe una carencia de herramientas que informen a los interesados en tiempo real de ciertos eventos o de cuándo se cumplen ciertas precondiciones.
_fuente: Heindl *et al.* [133]

Salvaguarda: S28.3 Se deberá utilizar una herramienta que facilite la configuración de alarmas para informar a los distintos roles de la satisfacción de ciertas precondiciones.
_controla: R38, R38.3
_fuente: Heindl *et al.* [133]

Clientes

Riesgo: R49 En el DGS puede ser difícil identificar los usuarios clave si los analistas no tienen acceso a la mayor parte del personal en su lugar de trabajo, y por tanto no pueden observar las tareas que los usuarios llevan a cabo.
_controla: Hanisch y Corbitt [130]

Salvaguarda: S29 Antes del comienzo del proyecto, clientes y proveedores deberán firmar un conjunto de condiciones que garanticen la implicación del cliente en el desarrollo del producto, en particular en el proceso de ingeniería de requisitos.
_controla: R41, R41.1, R41.2, R41.3, R43.1, R45, R46, R46.1, R47, R48, R49, R49.1, R49.2
_fuente: propuesta

Miscelánea

Riesgo: R58 En el DGS los desarrolladores se dan cuenta con frecuencia de que la motivación de los requisitos no está clara, no saben porqué los requisitos están formulados como están, de manera que es bastante común no comprender bien los requisitos, provocando retrasos y requiriendo un esfuerzo adicional para explicar los requisitos.
_fuente: Heindl y Biffl [132]

Salvaguarda: S32 Un atributo deberá estar incluido en el requisito para explicar su motivación, justificando la manera en la que el requisito esté formulado.
_controla: R58
_fuente: Heindl y Biffl [132]

Tabla 22 Distintos ejemplos del catálogo de amenazas y salvaguardas en el DGS

2.10.4.1 El modelo de procesos

El modelo de procesos de SIREN mostrado en la Sección 2.5 no es suficiente cuando se desea trabajar en un ambiente global. Se hace necesario por tanto extender dicho modelo de procesos con la adición de nuevas tareas, junto con un nuevo conjunto de roles encargados de desempeñarlas. El modelo de procesos que se propone para SIRENgsd combina un conjunto secuencial de tareas iniciales con otras cíclicas que se desarrollarían en espiral y que son básicamente las originales de SIREN recogidas en la Sección 2.5.2, que aquí hemos simplificado para centrar la discusión en las cuestiones relativas al DGS. En la Figura 14 se muestra gráficamente el proceso propuesto, que se centra en SIREN con reutilización y extiende el de la Figura 6.

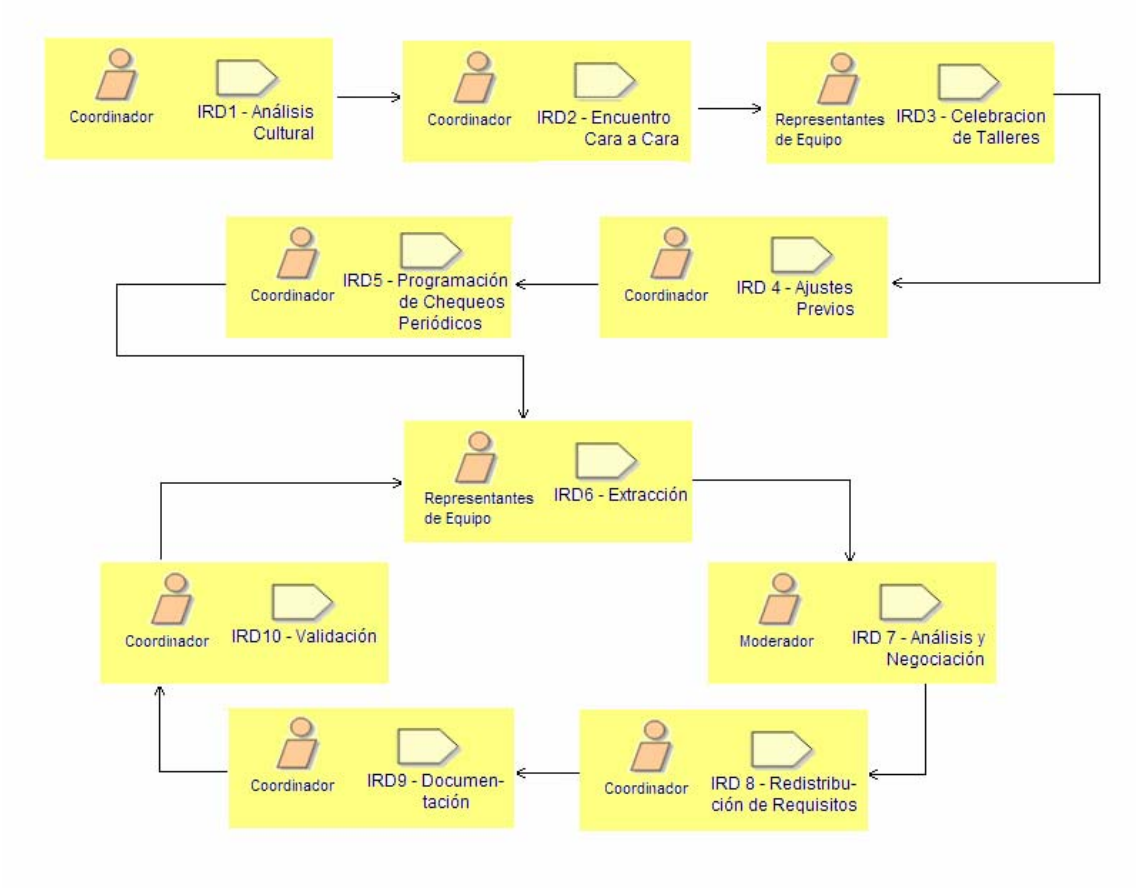

Figura 14 Modelo de procesos de SIRENgsd con la notación SPEM

Las tareas IRD6, IRD7, IRD9 e IRD10 se trasladan desde el modelo original de SIREN, mientras que IRD1–5 e IRD8 son totalmente nuevas. Las cinco primeras tareas son de preparación para que el proceso distribuido pueda llevarse a cabo con éxito, por tanto sólo tendrán que realizarse en la primera iteración. Las cinco últimas tareas conformarían la ejecución efectiva del proceso de IR y se realizarían de forma cíclica durante tantas iteraciones como fuera necesario. En la Tabla 23 se muestra un resumen de las tareas de SIRENgsd y en la Tabla 24 se muestran la entradas y salidas de cada tarea de forma esquemática.

El catálogo de buenas prácticas presentado en la Sección 2.10.3 ha servido de base para el diseño de este modelo de procesos. Las salvaguardas propuestas en el catálogo han

sido utilizadas para detallar las *subtareas* que tendrían que llevarse a cabo en cada una de las tareas para así salvar los riesgos o amenazas identificados. En SPEM se habla de pasos (*steps*) en la descripción de una tarea, pero preferimos introducir un nivel de abstracción más alto y tratar con lo que denominamos "subtareas", que a su vez conforman tareas. En [185] se trazan las tareas y subtareas del modelo de procesos a los riesgos y salvaguardas del catálogo.

Antes de estudiar cada tarea es necesario introducir los roles que intervienen en el modelo de procesos (Tabla 25), junto con su participación en las tareas (Tabla 26), una breve descripción de sus responsabilidades (Tabla 27) y un resumen de su participación en las tareas y subtareas (Tabla 28). En las responsabilidades de la Tabla 27 se hace referencia a las tareas y subtareas de SIRENgsd en las que cada rol está involucrado.

Tarea IRD 1. Análisis cultural

En esta tarea se deben analizar las distintas culturas de los participantes del proyecto, dado que las diferencias culturales se han identificado en el catálogo de riesgos y salvaguardas presentado en la Sección 2.10.3 como un problema que puede afectar al buen desarrollo de la IR distribuida. La información de entrada de la tarea son las nacionalidades de los miembros de los grupos de trabajo que participarán en el desarrollo. La salida de la tarea consiste en un informe en el que se detallan las diferencias culturales esenciales, que deberán ser tenidas en cuenta para facilitar las relaciones profesionales y evitar deterioros en las mismas, debidos a malentendidos o a falta de comprensión de la cultura con la cuál se trabaja.

Para realizar el análisis propuesto, resulta útil tener en cuenta cuatro dimensiones culturales, que han sido revisadas por López y Nicolás [185]:

- Dimensiones de Hofstede
- Dimensiones culturales de Hall
- Comunicación de alto contexto frente a la comunicación de bajo contexto
- Culturas universalistas y particularistas

Remitimos a los apéndices de [185] para obtener una descripción detallada de estas cuatro dimensiones culturales. Durante la explicación en este apartado se muestran someramente las dimensiones de Hofstede. A partir de estas teorías se extrae un conjunto de dimensiones o aspectos de interés que deben ser estudiados para cada cultura participante en el proyecto. El rol a cargo de realizar este estudio será el del *Coordinador*. Estos informes pasan a engrosar un Catálogo de Descripciones Culturales, de modo que puedan ser reutilizados.

Las subtareas que tendría que ejecutar el Coordinador dentro de esta tarea son las siguientes:

IRD 1.1 Identificación de nacionalidades. Se elabora un listado con las nacionalidades de todos los grupos de trabajo de los participantes en el proyecto, entendiendo por nacionalidad la cultura nacional en que cada persona ha crecido y a partir de la cuál ha adquirido su escala de valores. Si la granularidad de grupo de trabajo fuera inapropiada por demasiado grande se bajaría al nivel de persona.

Tarea	Subtareas	Descripción
IRD 1:Análisis cultural	IRD 1.1: Identificación de nacionalidades IRD 1.2 Recuperación de informes IRD 1.3 Elaboración de informes IRD 1.4: Mejora del Catálogo de Descripciones Culturales	Análisis de las diferencias culturales entre las culturas de los distintos países que intervienen en el proyecto.
IRD 2: Encuentro cara a cara	IRD 2.1 Encuentro cara a cara	Celebración de un encuentro cara a cara entre los participantes del proyecto que permita establecer las bases para forjar una relación de confianza y aliente la comunicación informal entre ellos.
IRD 3: Celebración de talleres	IRD 3.1 Celebración de talleres	Puesta en común dentro de cada grupo de trabajo de los informes culturales para conocer los aspectos a tener en cuenta cuando se trate con alguien que procede de una cultura diferente.
IRD 4: Ajustes previos	IRD 4.1: Identificación de Usuarios Clave IRD 4.2: Asignación de Usuarios Clave a equipos de trabajo IRD 4.3: Definición de un idioma oficial IRD 4.4: Compilación de un vocabulario común	Ajustes previos al comienzo del proyecto: identificación de Usuarios Clave, asignación de éstos a los grupos de trabajo, definición de un idioma oficial para el proyecto y construcción de una ontología que sea usada como glosario de términos para el proyecto.
IRD 5: Programación de chequeos periódicos	IRD 5.1: Programación de chequeos periódicos	Programación de chequeos periódicos de progresos en los que los grupos deberán informar a los otros sobre la evolución del trabajo.
IRD 6: Extracción	IRD 6.1: Selección de requisitos reutilizables IRD 6.2: Documentación de requisitos	Reutilización y extracción de requisitos.
IRD 7: Análisis y negociación	IRD 7.1: Preparación de la reunión IRD 7.2: Desarrollo de la reunión IRD 7.3: Extracción de conclusiones	Eliminación de inconsistencias del documento de requisitos mediante la discusión remota de distintos grupos de trabajo.
IRD 8: Redistribución de requisitos	IRD 8.1: Propuesta de asignación IRD 8.2: Validación de la asignación	Redistribución a los distintos grupos de trabajo de los requisitos extraídos y reutilizados para su desarrollo y refinamiento.
IRD 9: Documentación	IRD 9.1: Formalización de documentación	Establecimiento de los documentos de líneas base de requisitos.
IRD 10: Validación	IRD 10.1: Validación de requisitos	Validación con el cliente de los documentos de requisitos.

Tabla 23 Descripción de tareas de SIRENgsd

IRD 1.2 Recuperación de informes. Consulta del Catálogo de Descripciones Culturales y extracción de los informes ya existentes para las culturas con las que se trabaja.

IRD 1.3 Elaboración de informes. Elaboración de informes para las culturas que no se encuentran en el Catálogo de Descripciones Culturales. Se tendrán en cuenta las cuatro dimensiones culturales antes indicadas. Además se revisan los informes que han sido

extraídos del catálogo con el objetivo de encontrar errores y refinarlos si fuera posible. Una propuesta de plantilla para los informes culturales se muestra en la Tabla 29. Las dimensiones de Hofstede, que en IRD 3.1 se usan para comparar cómo de diferentes son dos culturas, pueden ser completadas consultando el valor del índice correspondiente [185]. El resto de los aspectos culturales no son tan precisos y objetivos, sino que tendrán que ser completados a partir de conversaciones con representantes de la cultura bajo estudio. Los campos referentes a la revisión aparecerán repetidos tantas veces como revisiones se hayan hecho del informe.

IRD 1.4 Mejora del Catálogo de Descripciones Culturales. Mejora del Catálogo de Descripciones Culturales mediante la adición de los informes que han sido elaborados y de los ya existentes que han sido mejorados.

Tarea	**Entrada**	**Salida**
IRD 1:Análisis cultural	Nacionalidades de los integrantes de los distintos grupos de trabajo participantes en el proyecto.	Informes culturales referentes a todos los países participantes en el proyecto.
IRD 2: Encuentro cara a cara	Listado de los participantes en el proyecto.	Se comienza a establecer una relación de confianza entre los participantes remotos en el proyecto.
IRD 3: Celebración de talleres	Informes culturales sobre los integrantes del proyecto.	Los miembros de los grupos de trabajo conocen las diferencias culturales con otros equipos de trabajo.
IRD 4: Ajustes previos	Información sobre las localizaciones de los grupos de trabajo e idiomas dominados por sus integrantes.	Listado de los Usuarios Clave junto con sus localizaciones y los grupos de trabajo que colaborarán con ellos, el idioma oficial del proyecto y el establecimiento de una ontología que modele el dominio de aplicación.
IRD 5: Programación de chequeos periódicos	Diferencias horarias entre los grupos de trabajo.	Programación de los informes de progresos de unos equipos de trabajo a otros.
IRD 6: Extracción	Repositorio de requisitos reutilizables, Usuarios y Usuarios Clave.	Documento de requisitos, provenientes del repositorio y de los usuarios, que puede contener inconsistencias. Orden del día con las cuestiones que hay que discutir sobre requisitos conflictivos.
IRD 7: Análisis y negociación	Orden del día para la reunión.	Documento de requisitos sin inconsistencias. Actas de la reunión en las que se detallan los aspectos tratados, los registros completos de las conversaciones mantenidas y los resultados de las votaciones llevadas a cabo.
IRD 8: Redistribución de requisitos	Documento de requisitos sin asignaciones, sólo con indicaciones de qué grupo de trabajo extrajo cada requisito.	Documento de requisitos con asignaciones del tipo grupo de trabajo – conjunto de requisitos.
IRD 9: Documentación	Documento de requisitos sin inconsistencias en desarrollo	Documento de requisitos en forma de línea base
IRD 10: Validación	Documento de requisitos pendiente de validación	Documento de requisitos validado y lista de modificaciones a realizar sobre el mismo

Tabla 24 Entradas y salidas de las tareas de SIRENgsd

Rol	Descripción
Coordinador	Encargado de la coordinación del trabajo de todos los participantes en el proyecto
Moderador	Modera las reuniones de negociación de requisitos
Representante de Equipo	Representa a su grupo de trabajo y habla en su nombre con el Coordinador y otros Representantes de Equipo
Analista	Desempeña labores de analista o ingeniero de requisitos propias de cualquier proyecto de desarrollo de software no distribuido
Usuario Clave	Conoce bien todo el sistema o una parte del mismo y aporta el conocimiento necesario para elaborar los documentos de requisitos.
Usuario	Conoce parte del sistema y aporta conocimiento necesario para elaborar los documentos de requisitos.

Tabla 25 Resumen de roles del catálogo de buenas prácticas

Tarea	Roles implicados	Responsable
IRD 1:Análisis cultural	Coordinador	Coordinador
IRD 2: Encuentro cara a cara	Coordinador, Moderador, Representantes de Equipo, Analistas	Coordinador
IRD 3: Celebración de talleres	Representantes de Equipo, Analistas	Representantes de Equipo
IRD 4: Ajustes previos	Coordinador, Usuarios, Usuarios Clave	Coordinador
IRD 5: Programación de chequeos periódicos	Coordinador	Coordinador
IRD 6: Extracción	Representantes de Equipo, Analistas, Moderador	Representantes de Equipo
IRD 7: Análisis y negociación	Moderador, Representantes de Equipo, Analistas (a través de sus Representantes de Equipo)	Moderador
IRD 8: Redistribución de requisitos	Coordinador, Moderador, Representantes de Equipo, Analistas (a través de sus Representantes de Equipo)	Coordinador
IRD 9: Documentación	Coordinador	Coordinador
IRD 10: Validación	Coordinador, Usuarios Clave	Coordinador

Tabla 26 Tareas y roles implicados

Tarea IRD 2. Encuentro cara a cara

Aún contando con herramientas como la videoconferencia, sigue siendo esencial realizar al menos un encuentro cara a cara al comienzo del proyecto [40, 41, 73, 80, 143, 252]. No sólo es necesario por ser una forma de comunicación más rica y eficiente que cualquier otra, sino porque es necesario para establecer una relación de confianza entre los participantes de un proyecto distribuido. Es muy difícil confiar en alguien con quién sólo se han establecido relaciones través de medios de comunicación electrónicos y no ha habido contacto informal [41, 80, 252].

Esta tarea consta de una sola subtarea:

Rol	Responsabilidades
Coordinador	- Estudio de las distintas culturas involucradas en el proyecto y mejora del Catálogo de Descripciones Culturales (IRD 1.4). - Organización y dirección de un encuentro cara a cara previo al comienzo del proyecto (IRD 2.1). - Delimitación de la duración de las tareas (en todas las tareas). - Identificación de Usuarios y Usuarios Clave (IRD 4.1). - Asignación de usuarios a grupos de trabajo (IRD 4.2). - Programación de las reuniones diarias de chequeo de progresos (IRD 5.1). - Elaboración de la propuesta de asignación de requisitos a grupos de trabajo (IRD 8.1). - Negociación con los Representantes de Equipos de la propuesta de asignación de requisitos a grupos de trabajo (IRD 8.2). - Publicación de la lista validada de asignación de requisitos a grupos (IRD 8.2). - Congelación de una versión estable de los documentos de requisitos y exportación a un formato que permita su validación (IRD 9.1). - Negociación personal con el cliente de los aspectos a modificar del documento de requisitos (IRD 10.1). - Elaboración y publicación de un documento de cambios a realizar en requisitos por petición expresa del cliente (IRD 10.1).
Representante del equipo	- Comunicación con otros Representantes de Equipos (durante todas las iteraciones). - Comunicación con el Coordinador (durante todas las iteraciones). - Organización y dirección de talleres culturales dentro de su equipo de trabajo (IRD 3.1). - Creación/mejora de la ontología que modele los conceptos del sistema, dominio o perfil (IRD 4.4). - Informe de progresos a equipos situados en otros husos horarios (durante todas las iteraciones). - Reutilización de requisitos (IRD 6.1). - Extracción de requisitos (IRD 6.2). - Redacción del documento de requisitos (IRD 6). - Adición de nuevos hilos de discusión al orden del día de las reuniones de negociación (IRD 6). - Estudio del orden del día de las reuniones de negociación (IRD 7.1). - Participación activa en las reuniones de negociación en las que se discuta sobre algún requisito en que ha trabajado algún miembro de su grupo de trabajo (IRD 7.2). - Representación y defensa de las opiniones de los miembros de su grupo de trabajo en las reuniones de negociación (textuales y mediante videoconferencia) (IRD 7.2). - Participación en las votaciones con la opinión acordada dentro del grupo de trabajo (IRD 7.2). - Negociación con el Coordinador de la propuesta de asignación de requisitos a grupos de trabajo (IRD 8.2).
Analista	- Creación/mejora de la ontología que modela los conceptos del sistema, dominio o perfil (IRD 4.4). - Reutilización de requisitos (IRD 6.1). - Extracción de requisitos (IRD 6.2). - Redacción del documento de requisitos (IRD 6). - Adición de nuevos hilos de discusión al orden del día de las reuniones de negociación (IRD 6). - Estudio del orden del día de las reuniones de negociación (IRD 7.1). - Labores propias de cualquier analista en un proyecto no distribuido (durante todas las iteraciones).
Moderador	- Revisión continua del documento de requisitos en busca de inconsistencias y errores (durante todas las iteraciones). - Preparación del orden del día de las reuniones de negociación (IRD 6). - Convocatoria de reuniones de negociación (IRD 6). - Intervenciones en las reuniones de negociación para mantener el orden (IRD 7.2). - Actualizar los progresos de las reuniones de negociación en la pizarra virtual de la herramienta de negociación (IRD 7.2). - Organización de votaciones (IRD 7.2). - Redacción y publicación de las actas de la reunión (IRD 7.3). - Modificación del documento de requisitos para reflejar los cambios acordados en la reunión (IRD 7.3).
Usuario	- Aporte de conocimiento sobre parte del sistema, dominio o perfil para la extracción de requisitos (IRD 6.2). - Validación de requisitos (IRD 10.1).
Usuario clave	- Aporte de conocimiento para la extracción de requisitos (IRD 6.2). - Validación de requisitos (IRD 10.1).

Tabla 27 Roles y responsabilidades de SIRENgsd

Coordinador	Moderador	Representante de equipo	Analista	Usuario clave	Usuario
IRD 1.1					
IRD 1.2					
IRD 1.3					
IRD 1.4					
IRD 2.1	IRD 2.1	IRD 2.1	IRD 2.1		
		IRD 3.1	IRD 3.1		
IRD 4.1					
IRD 4.2					
IRD 4.3					
IRD 4.4		IRD 4.4	IRD 4.4		
IRD 5.1					
		IRD 6.1	IRD 6.1		
	IRD 6.2	IRD 6.2	IRD 6.2	IRD 6.2	IRD 6.2
	IRD 7.1	IRD 7.1	IRD 7.1		
	IRD 7.2	IRD 7.2	IRD 7.2		
	IRD 7.3				
IRD 8.1					
IRD 8.2	IRD 8.2	IRD 8.2	IRD 8.2		
IRD 9.1					
IRD 10.1				IRD 10.1	

Tabla 28 Diagrama de calles para roles

IRD 2.1 Encuentro cara a cara. El Coordinador debería organizar un encuentro para todos los participantes en el proyecto. En principio no sería esencial que acudieran los clientes, aunque sería recomendable que al menos acudiera un representante del cliente, así podría conocer el equipo de trabajo y ver cómo afrontan el proyecto; se ha detectado que uno de los posibles problemas es que en ocasiones el cliente tiene la convicción de que el trabajo no puede realizarse desde una localización remota [272]. Idealmente todos los miembros de todos los grupos de trabajo tomarían parte en estas jornadas. Si no fuese posible, al menos sí deberían participar representantes de todos los equipos. Idealmente estas jornadas tendrían lugar en una localización atractiva, buscando dentro de lo posible la minimización de los costes de desplazamiento. Las jornadas deberían durar el tiempo necesario para que además de discutir las cuestiones iniciales sobre el proyecto, los participantes lleguen a conocerse un mínimo. Se organizarían diversas actividades que fomenten la interacción social y en las cuales personas que hasta ese momento no se conocían tengan que colaborar y ayudarse para lograr un objetivo común.

Tarea IRD 3. Celebración de talleres

Los grupos de trabajo deben organizar talleres de forma local, para estudiar las diferencias del resto de las culturas participantes con la suya propia. Los encargados de organizar estos talleres y distribuir la información dentro del grupo serían los Representantes de Equipo, que actúan como cabeza visible dentro del equipo y se comunican con otros Representantes de Equipo y con el Coordinador. Al final de esta tarea, los integrantes de los equipos (tanto Analistas como Representantes de Equipo) conocerán las diferencias culturales más importantes entre su propia cultura y cada una del resto de las culturas participantes en el proyecto. Además el Representante de Equipo deberá informar al Coordinador de que la tarea ha concluido y que todos los miembros de su equipo conocen los aspectos de otras culturas que podrían afectar al desarrollo del proyecto. Una vez que el Coordinador ha recibido la confirmación de todos los Representantes de Equipo, la tarea concluye y da comienzo la siguiente, IRD4.

Autor: *Fecha de elaboración:* *Revisado por:* *Fecha de revisión:* *…* *Cultura objeto de estudio:* *Idioma(s) nativo(s):* *Breve reseña histórica sobre el país:* *Breve descripción de la situación actual política, económica y social del país:* *Dimensiones culturales de Hofstede:* *Distancia al poder:* *Individualismo:* *Masculinidad:* *Elusión de incertidumbre:* *Orientación a largo plazo:* *Dimensiones culturales de Hall:* *Espacio:* *Bienes materiales:* *Amistad:* *Tiempo:* *Nivel de acuerdo:* *Monocronía/Policronía:* *Otras teorías:* *Cultura de alto/bajo contexto:* *Cultura universalista/particularista:*

Tabla 29 Propuesta de plantilla de un informe cultural

Esta tarea consta de una sola subtarea:

IRD 3.1 Celebración de talleres. En los talleres se deberá comparar una a una la(s) cultura(s) del grupo con el resto de culturas de los participantes en el proyecto. Un ejemplo del trabajo que debería desarrollarse en esta tarea puede verse en [167], donde se muestra una sencilla herramienta que permite comparar dos países según los índices de Hofstede, muestra gráficamente los resultados de la comparación y proporciona una serie de recomendaciones para tratar con culturas con niveles muy diferentes de la cultura de origen (ver Tabla 30 y Tabla 31).

Leyenda	
O < T	La cultura de procedencia tiene un valor menor en el índice en cuestión que la cultura en la que se trabajará
O > T	La cultura de procedencia tiene un valor mayor en el índice en cuestión que la cultura en la que se trabajará

Tabla 30 Leyenda para interpretar las recomendaciones de la Tabla 31

Tarea IRD 4: Ajustes previos

Esta tarea tiene como objetivo realizar los últimos ajustes previos a la extracción de requisitos y comprende subtareas de (1) Identificación de Usuarios Clave; (2) Asignación de equipos; (3) Definición de un idioma oficial; y (4) Compilación de un vocabulario común. Estas subtareas se repasan a continuación:

Distancia al poder	0 < T	- Imparte órdenes claras y explícitas a los que trabajan contigo. Los plazos de entrega tienen que ser bien señalados. - No esperes que los subordinados tomen la iniciativa. - Se más autoritario en tu estilo de dirección. Las relaciones con el personal pueden ser más distantes de lo que estás acostumbrado. - Muestra respeto y deferencia a aquellos más altos en la jerarquía. Normalmente esto se muestra a través del lenguaje, comportamiento y protocolo. - Espera encontrar más burocracia en agencias y organizaciones gubernamentales.
	0 > T	- No esperes ser tratado con el respeto o la deferencia a los que estás acostumbrado. - La gente querrá conocerte de un modo informal sin mucho protocolo o etiqueta. - Intenta ser más inclusivo en tu estilo de dirección o liderazgo, ya que ser muy dictatorial puede ser mal interpretado. - Involucra a los demás en la toma de decisiones. - No juzgues a las personas basándote en apariencia, comportamiento, privilegios o símbolos de estatus.
Individualismo	0 < T	- Recuerda que no puedes depender del grupo para obtener respuestas. Como un individuo se supone que tienes trabajar tú solo y usar tu iniciativa. - Prepárate para un entorno de negocios que puede confiar menos en relaciones y contactos personales. Los negocios y la vida personal pueden mantenerse separados. - Los empleados o subordinados esperan la oportunidad de trabajar en proyectos o resolver cuestiones de forma independiente. - Entrometerse demasiado en su trabajo puede interpretarse negativamente. - No es extraño que la gente intente sobresalir respecto de los demás durante reuniones, presentaciones o durante el trabajo en grupo. - Ten en cuenta que se tolera una cierta cantidad de expresión individual, por ejemplo, la apariencia o el comportamiento.
	0 > T	- Ten en cuenta que los individuos pueden tener un fuerte sentido de la responsabilidad hacia su familia, lo que quiere decir que le darán prioridad frente a los negocios. - Recuerda que se debe elogiar al equipo nunca al individuo, ya que esto podría causar vergüenza. Recompensa a los equipos no a la persona. - Entiende que la promoción depende del tiempo en la empresa y de la experiencia, no del rendimiento o de los logros. - La toma de decisiones puede ser un proceso lento, ya que muchos individuos de la jerarquía deben ser consultados.
Masculinidad	0 < T	- Para tener éxito en esta cultura se espera que hagas sacrificios en la forma de más horas de trabajo, vacaciones más cortas y posiblemente más viajes. - Ten en cuenta que la gente hablará de negocios todo el tiempo, incluso en encuentros sociales. - Evita preguntar cuestiones personales en situaciones de negocios. Tus colegas querrán probablemente ir directos al grano. - La gente no está siempre interesada en desarrollar una profunda amistad. - El estilo de comunicación directo, conciso y carente de emociones será más efectivo en este entorno. - La gente utiliza la identidad profesional, preferiblemente a la familia o contactos para evaluar a otros. - La auto promoción es una parte aceptable de la cultura de negocios en este entorno competitivo.
	0 > T	- Ten en cuenta que la gente valora su tiempo personal. Dan prioridad a su familia y toman vacaciones más largas. Las horas extra no son la norma. - La charla en situaciones sociales o de negocios se centrará en la vida individual y en intereses más que solamente en negocios. - Las preguntas personales son normales, no molestas. - En los negocios, la confianza tiene más peso que los márgenes de beneficio y similares. - El nepotismo se ve como algo positivo y la gente muestra abiertamente favoritismo hacia relaciones cercanas.
Elusión de incertidumbre **(inquietud ante lo desconocido)**	0 < T	- No esperes que nuevas ideas, formas o métodos sean rápidamente aceptados. Tendrás que dejar tiempo para que ayude a desarrollar un entendimiento de la iniciativa para estimular la confianza en ella. - Involucra a los locales en proyectos para que los comprendan. Esto disminuirá el elemento desconocido. - Prepárate para una visión del mundo más fatalista. La gente puede no sentirse totalmente segura si no se encuentran en una situación conocida y por tanto controlada, así que posiblemente estarán menos dispuestas a tomar decisiones con algunos elementos desconocidos. - Recuerda que debido a la necesidad de negar la incertidumbre, las proposiciones y presentaciones serán examinadas con lupa. Respalda todo lo que digas con hechos y estadísticas.
	0 > T	- Intenta ser más flexible o ábrete a nuevas ideas más de lo que solías estar. - Prepárate a poner en marcha planes acordados rápidamente ya que se esperará que se lleven a cabo tan pronto como sea posible. - Da a los empleados autonomía y espacio para que ejecuten sus tareas por ellos mismos. Sólo se espera de ti directrices y recursos. - Acepta que los habitantes del país pueden tener una aproximación distinta a la vida y ver su destino en sus propias manos.

Tabla 31 Recomendaciones para tratar con culturas con diferentes valores en los índices de Hofstede

IRD 4.1 Identificación de Usuarios Clave. Hanish y Corbitt [130] describen un problema que puede aparecer si no se distinguen adecuadamente Usuarios y Usuarios Claves: se extraen ciertos requisitos a partir de algunos usuarios, se refinan y documentan. A la hora de validarlos, algunos Usuarios Clave dicen que no son correctos, que faltan muchos requisitos importantes, y los que están tienen que sufrir

profundas remodelaciones. Así pues antes de que comience la extracción de requisitos se deben identificar las personas que juegan los roles de Usuario y Usuario Clave.

El problema de la identificación de Usuarios Clave también se da en la IR colocalizada, pero al desplazarnos a un entorno global, el problema se vuelve más complicado, ya que la distancia y la imposibilidad de observar a los usuarios desempeñando su labor habitual puede dificultar su correcta identificación.

IRD 4.2 Asignación de Usuarios Clave a equipos de trabajo. Dependiendo de la localización de los equipos de trabajo y de los Usuarios Clave, a cada equipo le corresponde un conjunto de Usuarios Clave para extraer requisitos. Idealmente cada dupla "equipo de trabajo–usuario clave" debería encontrarse colocalizada, de este modo la distancia no condicionaría la técnica de extracción de requisitos a emplear. Si fuera inevitable que un grupo de trabajo tuviera a sus Usuarios Clave físicamente distantes, tendría que utilizar técnicas de extracción de requisitos compatibles con los medios de comunicación electrónica.

Idealmente sólo los Usuarios Clave deberían colaborar en la extracción de requisitos. En caso de que esto no fuera posible, intervendrían también usuarios no clave, pero a la hora de la asignar prioridades y negociar requisitos, en caso de conflicto, se daría más importancia a los requisitos provenientes de Usuarios Clave.

IRD 4.3 Definición de un idioma oficial. Aunque en las interacciones sociales informales todos los roles utilicen el idioma en que se sientan más cómodos, es necesario definir un idioma oficial para el proyecto. Este lenguaje común será el utilizado en cualquier reunión formal de negociación de requisitos u otra situación en que intervengan participantes de distintas nacionalidades e idiomas maternos. El Coordinador define el idioma común basándose en la información sobre las nacionalidades de los grupos de trabajo.

IRD 4.4 Compilación de un vocabulario común. En cualquier proyecto de desarrollo de software es necesaria la elaboración de un glosario de términos que elimine ambigüedades. En el caso en que no todos los equipos de trabajo compartan el mismo idioma materno, la situación se agrava: incluso dentro de la misma lengua, hay diferencias de vocabulario entre países o incluso regiones. Con el objetivo de obtener un corpus de conceptos homogéneo para todos los grupos de trabajo con independencia de su origen cultural, Aranda *et al.* [28] proponen el uso de una *ontología* para solucionar este problema. Una ontología es un esquema conceptual riguroso y exhaustivo dentro de un dominio dado con la finalidad de facilitar la comunicación y compartición de información entre sistemas.

Un Catálogo de Ontologías almacena las ontologías que han sido utilizadas en algún proyecto. IRD4.4 consiste en la extracción de una ontología relevante del Catálogo de Ontologías y su refinamiento mediante la adición de conceptos y relaciones durante todo el proceso de extracción de requisitos. Si no existiera una ontología para el dominio vertical u horizontal implicado habría que construirla desde cero. Puesto que la ontología debe traducir los conceptos a todos los idiomas de los participantes (idioma común incluido), cada equipo de trabajo debería ampliarla no sólo con los nuevos conceptos que extraiga de la interacción con los usuarios, sino también con las traducciones de los conceptos que utilice a su(s) idioma(s).

Tarea IRD 5. Programación de chequeos periódicos

La distancia en los proyectos de desarrollo global de software conlleva muchos problemas, pero también permite un modelo de trabajo *follow-the-sun* [41] o *around-the-clock* [77, 132, 134], es decir permite trabajar "siguiendo el sol" o "alrededor del reloj", aprovechando las diferencias horarias para conseguir jornadas laborales globales de 24 horas. Para ver un ejemplo se puede acudir al caso de estudio presentado por Hanisch y Corbitt [130].

En esta tarea hay una única subtarea:

IRD 5.1 Programación de chequeos periódicos. El Coordinador debe estudiar las localizaciones de los distintos grupos de trabajo y sus diferencias horarias para programar reuniones diarias de forma que puedan aprovecharse al máximo los husos horarios. Estas reuniones son necesarias además para llevar un seguimiento del trabajo del resto de equipos, ya que se ha detectado que se tiende a asumir que el trabajo en localizaciones remotas va bien [40], lo cual puede provocar que el proyecto fracase. En muchas ocasiones no será sencillo ajustar los horarios y las reuniones serán incómodas, pero de cualquier modo es importante que se programen estas reuniones para que todos los grupos conozcan cómo va el trabajo de los demás.

Tarea IRD 6. Extracción de requisitos

Los cambios que sufre esta tarea con respecto a la propuesta original de SIREN son mínimos. Las tareas que antes se llevaban a cabo en una única localización, ahora estarán repetidas en distintas localizaciones. Cada grupo de trabajo reutilizará y extraerá requisitos de forma local. Es posible que los requisitos también se deban extraer de forma distribuida: que los Analistas tengan que entrevistarse con usuarios pertenecientes a distintos departamentos distribuidos por el mundo. En este escenario, distintos grupos de Analistas extraerán requisitos de distintos grupos de usuarios, con el objetivo de conseguir un conjunto coherente de requisitos para el producto software completo, que después será utilizado por todos los usuarios.

Las subtareas comprendidas en esta tarea son las siguientes:

IRD 6.1 Reutilización de requisitos. Cada equipo de trabajo selecciona del catálogo de requisitos de SIREN aquellos perfiles que sean necesarios para el proyecto desde su punto de vista (la visión que tendrá cada grupo de trabajo es la que le aportará el grupo de usuarios a los que tenga acceso), instancia los requisitos parametrizados y los añade al documento de requisitos actual.

IRD 6.2 Extracción de requisitos del proyecto. Cada equipo de trabajo da forma a los requisitos que hayan obtenido de los usuarios a los que tienen acceso y los añade al documento de requisitos actual.

En el desarrollo global, como hay múltiples fuentes de requisitos, es más probable que en el desarrollo tradicional, por ejemplo, (1) que un mismo requisito aparezca con valores diferentes para un mismo parámetro; (2) que equipos distintos seleccionen requisitos que son exclusivos; o (3) que a través de las relaciones de trazas inclusivas se

dupliquen requisitos. La herramienta CARE de soporte debería evitar estos problemas. Esta herramienta debe permitir que múltiples usuarios editen a la vez el mismo documento, y por tanto cuando un usuario añada un requisito del catálogo, la herramienta mantendrá automáticamente todas las relaciones de traza, es decir, comprobará si tiene que añadir su requisito padre, si implica a otros requisitos o si viola alguna relación de exclusión con alguno de los requisitos previamente añadidos. Al hacer estas comprobaciones se garantiza en todo momento que los requisitos serán consistentes, ya que habrá un único documento de trabajo. Otra cuestión es que Analistas diferentes introduzcan el mismo requisito asignando distintos valores al mismo parámetro. En ese caso, cuando el segundo Analista intente introducir el requisito conflictivo en el documento, la herramienta deberá detectar la inconsistencia y notificarla a ambos. Los Analistas implicados habrán de discutir cuál es el valor correcto del parámetro y en caso necesario llevarán la cuestión al orden del día de la próxima reunión de negociación de requisitos.

Con respecto a los requisitos que no han sido reutilizados, los que han sido directamente extraídos, en principio serán añadidos al documento según el juicio de los propios Analistas. El Moderador revisará los requisitos añadidos por los distintos grupos de trabajo, recopilará todas las inconsistencias que encuentre y si fuera necesario las añadirá al orden del día de la próxima reunión. Cuando juzgue que tiene suficientes inconsistencias sin resolver, convocará a todos los equipos de trabajo a una reunión (IRD 7). Además cada Analista podrá añadir al orden del día todas aquellas cuestiones que considere que merecen ser discutidas, siempre añadiendo su identidad para facilitar las discusiones.

Así pues, al final de esta tarea se tendrá una jerarquía de documentos de requisitos (que coincide con la estándar de SIREN) coherente con respecto a los requisitos reutilizados, con la posible excepción de algunos parámetros y con requisitos extraídos provenientes de todos los grupos de trabajo, en los que es posible que haya inconsistencias. Además existirá un orden del día que detallará las cuestiones que deberán ser tratadas en la próxima reunión de negociación de requisitos.

Tarea IRD 7. Análisis y negociación

En esta tarea se parte de un documento de requisitos con inconsistencias, en el que hay cuestiones por discutir, con el objetivo de alcanzar una versión refinada del mismo, en el que todas las inconsistencias hayan sido resueltas. Al concluir esta tarea los documentos de requisitos habrán sido actualizados para eliminar todas las incoherencias y estarán disponibles las actas de la reunión en la herramienta de trabajo.

Esta tarea consta de las siguientes subtareas:

IRD 7.1 Preparación de la reunión. Al concluir IRD 6 debe existir un orden del día con las cuestiones que deberán ser discutidas en la siguiente reunión de negociación de requisitos, que habrá sido elaborado a partir de tres fuentes: las inconsistencias detectadas automáticamente por la herramienta en los valores parametrizados, cuestiones sugeridas por Analistas y conflictos detectados por el moderador.

Este orden del día de la reunión deberá ser leído antes del comienzo del encuentro por todos los participantes que hayan sido convocados, es decir, aquellos grupos

involucrados en un conflicto a causa de un parámetro, los grupos responsables de un requisito que debe ser discutido a juicio del moderador y los grupos que propusieron un hilo de discusión. El orden del día estará accesible en todo momento en la herramienta de trabajo.

IRD 7.2 Desarrollo de la reunión. El soporte sobre el cuál la discusión se llevará a cabo será una parte de la herramienta de trabajo que consistirá en un sistema de comunicación textual síncrono estructurado. La labor del Moderador será intervenir cuando lo considere necesario para imponer orden y mantener actualizados los progresos de la reunión en un pizarra virtual, indicando qué puntos del orden del día ya han sido tratados, con qué resultados y qué se trata actualmente. Para facilitar la discusión sólo los Representantes de los Grupos participarán hablando en nombre de sus equipos. Es posible que para algunas cuestiones se concluya que lo más razonable sea votar una solución, se empleará entonces una utilidad de votación que permitirá elegir el sistema más conveniente en cada situación (ver Sección 2.10.4.3). Como regla general recomendamos que antes de comenzar una votación el Moderador seleccione un sistema de voto lo más sencillo posible (por ejemplo, una sola vuelta con mayoría absoluta) y sólo si la votación resulta ser más complicada de lo esperado, que elija un sistema más elaborado (por ejemplo, una primera vuelta con todas las posibilidades, y una segunda en que sólo se vote a las dos candidatos más populares de la primera vuelta).

Es de esperar que así la mayoría de los problemas hayan quedado resueltos, pero es posible que aún queden cuestiones pendientes. Una vez se ha discutido suficientemente de forma textual, puede ser apropiado resolver los últimos detalles usando un medio de comunicación más rico, la videoconferencia. Este medio es útil para salvar la distancia pero es difícil mantener una discusión larga cuando intervienen más de unas pocas personas: como afirman Damian *et al.* [79], resulta necesario que haya un trabajo previo. De ahí que en esta tarea la videoconferencia se utilice sólo en este momento: (1) los participantes han tenido tiempo de estudiar los puntos del día consultando toda la información que necesiten, sin la presión de una comunicación síncrona; (2) se discute mediante un medio síncrono textual controlado por el Moderador, de manera que al tener que escribir, obliga a que se mediten las intervenciones antes de hacerlas; (3) una discusión cara a cara, que permite evaluar mejor las reacciones de los demás basándose en lenguaje corporal, terminará de resolver cualquier cuestión no resuelta.

IRD 7.3 Extracción de conclusiones. El Moderador redacta las actas de la reunión, con los puntos discutidos y conclusiones alcanzadas, tanto en la comunicación textual como en la videoconferencia, además de los resultados de las votaciones que puedan haberse llevado a cabo, y publicará las actas en la herramienta de trabajo. Estas actas estarán asociadas a un código que las identificará unívocamente dentro del proyecto y a un objeto *Hilo de discusión* que almacenará el registro completo de las conversaciones mantenidas durante la negociación junto con los resultados de todas las posibles votaciones que se hayan celebrado. También reflejará en el documento de requisitos los cambios que se hayan acordado.

Con el proyecto fin de carrera de María Martínez [114], que recientemente se ha iniciado, se pretende realizar (1) una revisión sistemática de la literatura sobre negociación de requisitos aplicable al DGS; y (2) con base en el conocimiento del estado del arte adquirido en la etapa anterior, realizar una formulación más detallada de esta actividad IRD 7.

Tarea IRD 8. Redistribución de requisitos

Esta tarea no pertenece a la versión original de SIREN, y justifica su existencia dado el carácter distribuido de la extracción de requisitos en el DGS. A partir del documento de requisitos, que contendrá información acerca de qué grupo ha extraído cada requisito, se obtendrá una asignación de requisitos a grupos de trabajo, que no tiene necesariamente que coincidir con la asociación que había inicialmente.

Consideremos el concepto de *interfaces organizacionales* propuesto por Gumm [128]. Una interfaz organizacional se define como dos unidades organizacionales que interactúan en un proyecto, donde una unidad organizacional es un departamento de la organización, una organización completa, desarrollos de proyectos, niveles jerárquicos o roles. En este caso consideraremos interfaces organizacionales compuestas por un equipo de trabajo y el grupo de usuarios que tienen asignados. Distintos subconjuntos de requisitos han sido aportados (extraídos o reutilizados) por distintas interfaces organizacionales, pero no es obligatorio que se mantenga este statu quo, sería desaprovechar una de las ventajas del desarrollo global de software. Por ejemplo, es posible que un grupo haya extraído requisitos referentes a la seguridad del sistema porque tiene a su disposición usuarios que trabajan en temas de seguridad, mientras que en otro grupo de trabajo hay un Analista especialista en seguridad: probablemente lo adecuado en ese caso sería que el conjunto de requisitos referentes a seguridad fuera refinado y ampliado por el grupo al que pertenece el experto en seguridad.

Las subtareas que se llevan a cabo en esta tarea son las siguientes:

IRD 8.1 Propuesta de asignación. El Coordinador elaborará una propuesta de asignación de conjuntos de requisitos a grupos de trabajo tomando como criterios las habilidades más destacadas de cada equipo, la carga de trabajo, etc. Esta propuesta se pondrá a disposición de todos los grupos a través de la herramienta de soporte y se convocará una reunión para discutirla.

IRD 8.2 Validación de la asignación. Una vez que todos los equipos hayan tenido la oportunidad de estudiar y discutir la propuesta en profundidad, se celebrará una reunión virtual en la que participarán todos los Representantes de Grupo y el Coordinador y en la que se discutirán las cuestiones que hayan surgido en torno a las asignaciones. Si todos los Representantes están de acuerdo se validaría la distribución y se publicaría en la herramienta. Si hubiese desacuerdos, se procedería como en una reunión habitual (IRD 7). Al igual que en las reuniones de negociación de requisitos, la figura del Moderador será esencial, tendrá que regular la discusión, mantener la pizarra con los progresos de la reunión y levantar el acta al terminar. Tras la reunión se debe obtener una redistribución de requisitos validada, que será publicada en la herramienta de trabajo.

Tarea IRD 9. Documentación

Debido a la naturaleza del proceso y de la herramienta de trabajo, esta tarea será muy simple. Se partirá de un documento de requisitos, idealmente coherente y sin inconsistencias, que dependiendo de la iteración del proceso en que nos encontremos, será más o menos detallado. El resultado sería establecer ese documento como una línea

base, para que deje de ser modificado y pueda ser presentado a los usuarios para que lo validen. Esta tarea consta de una sola subtarea:

IRD 9.1 Formalización de documentación. El Coordinador utilizará una opción de la herramienta que permita crear una línea base a partir de una versión estable de la documentación, almacenarla en un catálogo de versiones y exportarla a un formato de documento estándar. Esta versión estable será la que se utilice en la siguiente tarea de validación, mientras que la versión actual seguirá evolucionando en otra iteración si el proceso no hubiera concluido.

Tarea IRD 10. Validación

Esta tarea será idéntica a la que se llevaría a cabo en un desarrollo no distribuido de software. Consta de una sola subtarea:

IRD 10.1 Validación de requisitos. El Coordinador entregará el documento de salida de la tarea anterior al cliente, negociará con él personalmente todos los cambios que solicite y transmitirá esta información a los Responsables de Grupo, mediante una lista de modificaciones que hará pública a través de la herramienta.

2.10.4.2 Las técnicas

En esta sección se revisan los cambios en las técnicas involucradas en el método SIREN cuando se desarrolla en el DGS y se proponen ampliaciones del metamodelo de requisitos de SIREN.

El conjunto mínimo de atributos de SIRENgsd

Para dar soporte a la reutilización en un ambiente distribuido de software, en este apartado se proponen atributos adicionales al conjunto mínimo de atributos de SIREN mostrado en la Sección 2.6.2:

- *solicitadoPor*: la semántica de este atributo sería la misma, pero el atributo pasaría a tener carácter forzoso, ya que posiblemente haya que recurrir al origen del requisito para clarificar cuestiones, puesto que es posible que el equipo que desarrolle el requisito no sea el mismo que lo extrajo.
- *grupoDeTrabajoFuente* (forzoso): identificador del grupo de trabajo que extrajo o reutilizó el requisito. Al igual que con el atributo anterior, es posible que el responsable del requisito necesite consultar a los Analistas que lo extrajeron.
- *analistaFuente* (forzoso): Analista que redactó o reutilizó el requisito, se debe saber quién es responsable de que el requisito aparezca en la documentación en la forma actual.
- *grupoDeTrabajoDestino*: hace referencia al identificador del grupo de trabajo al que se ha asignado el requisito para su ampliación o refinamiento. Este atributo estará vacío hasta la tarea IRD 7, “Redistribución de requisitos”.
- *estado*: se añade un nuevo estado a este atributo:

  - *Pendiente de discusión*: indica que hay inconsistencias entre grupos de trabajo en relación al requisito y que se debe discutir sobre el asunto en la tarea IRD 6, "Análisis y negociación".

- *hiloDeDiscusión*: hace referencia a los identificadores de las actas de reunión en las que se discutió sobre el requisito. Estas actas contienen un enlace a un objeto *Hilo de discusión* que contiene el registro de la conversación mantenida en la negociación de requisitos asociada, junto con los resultados de las posibles votaciones que surgieron en el seno de dicha reunión. De este modo será posible mantener la traza de los requisitos a las discusiones que suscitaron.

El metamodelo de trazabilidad de SIRENgsd

El metamodelo de trazabilidad de SIREN (Sección 2.6.4) se extiende con una nueva relación de traza:

- *Traza hiloDeDiscusión* (*discussionThread*): relación de un requisito y la reunión en la que fue discutido, a través de las actas de reunión y el objeto Hilo de discusión que contiene el registro completo de la discusión junto con las posibles votaciones realizadas.

Las plantillas de documentos de SIRENgsd

La estructura de documentos de SIREN (Sección 2.6.5) es compatible con un desarrollo distribuido de software. Sólo habrá que tener en cuenta que el soporte que contenga estas plantillas permita la edición concurrente y distribuida de requisitos. Seguramente habrá que ir a una herramienta de soporte web, como se indica en la siguiente sección.

2.10.4.3 La herramienta CARE

La herramienta de soporte es un aspecto esencial para conseguir que la aplicación de SIRENgsd sea factible. A partir de los escenarios típicos de ejecución del proceso, López y Nicolás [185] definen un conjunto inicial de 29 características (*features*) que debería tener la herramienta de soporte a SIRENgsd que extendiera SirenTool para su uso en el DGS, y se extienden en la descripción de un posible diseño para dicha herramienta. En este apartado nos limitamos a ofrecer un resumen muy breve de esta discusión.

La herramienta está concebida para mantener el conocimiento global del proyecto, para que en todo momento cada Analista sepa quién está trabajando en qué requisito mediante los atributos grupoDeTrabajoFuente, analistaFuente y grupoDeTrabajoDestino. Así se sabe qué personas hay que notificar ante un cambio en algún requisito, de manera que nadie trabajará sobre un requisito obsoleto.

El sistema debe fomentar la comunicación informal, de ahí que debería integrar utilidades de mensajería instantánea y videoconferencia. Al margen de propiciar una respuesta más rápida que con el uso del correo electrónico, este tipo de comunicaciones informales tienen la ventaja de que ayudan a afianzar las relaciones de confianza entre equipos de trabajo geográficamente alejados.

Como alternativa de diseño, proponemos seleccionar el uso de una wiki. No es una idea nueva, Decker *et al.* [84] proponen el uso de una wiki especialmente preparada para soportar la elaboración distribuida asíncrona de documentos de requisitos. Tras implementar un modelo de plantillas enlazadas entre sí apoyándose en funcionalidades clásicas de las wikis y añadiendo otras específicas, validaron esta propuesta en casos de estudio reales que según afirman quedaron satisfechas con el resultado obtenido y adoptaron la herramienta.

Con el proyecto fin de carrera de Mariano Madrigal [190] se ha iniciado recientemente el desarrollo de una herramienta CARE de soporte a SIRENgsd.

2.11 Conclusiones

En este capítulo se han recogido los trabajos relacionados con la línea inicial de trabajo de esta tesis doctoral, IR y reutilización. Se trata de la línea de investigación base a partir de la cual se derivan las dos posteriores, IR para líneas de productos (método SIRENspl, Capítulo 3), y Generación de especificaciones de requisitos textuales a partir de modelos de ingeniería del software del sistema (en el ámbito de SIRENspl, Capítulo 4). Con el objetivo de establecer un contexto para esta primera línea de investigación, se ha comenzado presentando una visión general del estado del arte en IR y en reutilización de requisitos en lenguaje natural. Después se ha presentado en detalle SIREN, incluyendo sus experiencias de aplicación y su herramienta de soporte, y finalmente se ha mostrado una extensión de SIREN para considerar el DGS.

2.11.1 La aportación al estado del arte

En el ámbito de la reutilización de requisitos, con SIREN se proporciona un método práctico, sencillo y flexible para mejorar el estado de la práctica de IR, que ha sido probado en entornos industriales y que pensamos puede servir como un primer paso para acercar dicho estado de la práctica al estado de la investigación en IR. Buscando flexibilidad, SIREN se concibe como una especie de "*plugin* metodológico" que señala un conjunto reducido de tareas, técnicas y guías que pensamos se pueden extrapolar al marco dado por cualquier método de IR. A partir de la experiencia ganada con SIREN se han identificado ocho cuestiones clave sencillas que en nuestra opinión deberían ser soportadas por cualquier método de IR basado en reutilización y que son de utilidad para evaluar un método o una herramienta de IR basadas en reutilización.

En términos de los campos utilizados en la síntesis de enfoques de reutilización de requisitos relacionados con lenguaje natural, presentada en la Tabla 11 de la página 63, nuestro enfoque SIREN se resume como se recoge en la Tabla 32.

Llegados a este punto destacamos las siguientes aportaciones de SIREN:

- Se trata del primer enfoque que conocemos [284] que abordó el problema de la reutilización de requisitos de seguridad y de protección de datos personales utilizando catálogos de requisitos en lenguaje natural especificados a partir de estándares (como la metodología MAGERIT, estándar de la Administración

española compatible con el *Common Criteria Framework*) y de textos legales (como la LOPD y el RMS).

- Se trata del primer enfoque que conocemos [177, 280] que proporcionó soporte automatizado a la reutilización a través de una herramienta CARE comercial (en particular, a través de IBM Racional Requisite Pro, aunque el enfoque es trasladable a otras herramientas comerciales). El soporte automatizado a la reutilización de requisitos sigue siendo escaso en la actualidad.
- Se trata de uno de los primeros enfoques [187] de IR en entornos globales de software, y el único que conocemos que considera explícitamente la reutilización.
- Se trata del único enfoque que conocemos [280] que proporciona guías explícitas para la reutilización de metainformación en forma de atributos y trazas. Estas cuestiones forman parte de una lista de ocho cuestiones clave para la reutilización, que se han determinado como resultado de la experiencia acumulada con este método.

Gen.	Com.	Modelo de procesos	Técnicas y guías	Organización de requisitos	Herramienta	Validación
■	■	Sí, modelo de procesos en espiral con adiciones para la reutilización (principalmente enfoque *con* reutilización)	Requisitos en lenguaje natural incluyendo parametrizados; modelo de referencia de metainformación, incluyendo atributos y trazas; guías de: definición de requisitos, atributos, requisitos parametrizados, uso de trazas, organización de catálogos de requisitos, reutilización de atributos y trazas, mejora del repositorio, organización de la documentación de requisitos	Catálogos de requisitos, por dominios y perfiles; búsqueda por palabras clave y metainformación; MM de requisitos explícito utilizando MOF	SirenTool, *plugin* sobre IBM Rational Requisite Pro	(CEI) Cinco casos de estudio (proyecto GARTIC); catálogos PDP (LOPD/RMS) y SEGURIDAD (MAGERIT); auditoría de seguridad informática
Leyenda: Enfoque: *Gen*. Basado en generación; *Com*. Basado en composición; LDP: Línea De Productos; MM: Metamodelo; OO: Orientado a Objetos; CEI: Caso de Estudio Industrial / CEA: Caso de Estudio Académico						

Tabla 32 SIREN en términos de la síntesis de enfoques de reutilización de requisitos de la Tabla 11

2.11.2 Conclusiones adicionales y vías futuras

Pensamos que los resultados planteados en este capítulo son de interés para profesionales del desarrollo de software, desarrolladores de herramientas CARE e investigadores: (1) los profesionales del desarrollo de software disponen de un método suficientemente sencillo como para ser adoptado en un corto espacio de tiempo por una organización inmadura en cuanto a su proceso de requisitos. En caso de que la organización sea madura en cuanto a la gestión de los requisitos y por tanto haya definido explícitamente su método de IR, con SIREN se dispone de un conjunto de tareas, técnicas y guías que se pueden incorporar a dicho método de IR para que soporte la reutilización de requisitos textuales en lenguaje natural, que siguen ocupando un lugar importante en la industria y que facilitan la participación de todos los interesados en el análisis, negociación y validación de los requisitos. SIREN se ha adoptado con éxito por parte de cinco pequeñas y medianas empresas de desarrollo en el proyecto GARTIC, y los analistas involucrados no lo han considerado como una sobrecarga de trabajo importante, al tiempo que consideran muy significativo el valor añadido de reunir en un catálogo de requisitos el conocimiento de la empresa sobre un dominio de aplicación en el que ha realizado desarrollos o tiene previsto iniciarlos; (2) los desarrolladores de herramientas CARE disponen de una serie de ocho cuestiones clave que sus herramientas deben soportar de una forma u otra. Se trata de cuestiones sencillas, que plantean cuestiones que creemos son fácilmente extrapolables al

metamodelo de las técnicas soportadas por cualquier herramienta de requisitos, sin crear problemas operativos reseñables; y (3) los investigadores disponen de un método sencillo y probado, basado en reutilización de requisitos, que pueden extender con trabajos en otras dimensiones: desarrollo global (como la extensión realizada en este capítulo, que todavía no se ha validado), IR basada en metas, trazabilidad y componentes, aspectos de análisis (*early aspects*), etc.

Tradicionalmente SIREN ha prestado más atención al desarrollo con reutilización que al desarrollo para reutilización. Las definiciones iniciales de SIREN [284, 285] lo planteaban como un método de IR con reutilización, considerando preexistentes los catálogos de requisitos en el catálogo. Esto es así porque los catálogos SIREN desarrollados inicialmente en el GIS, de protección de datos personales (Seguridad-PDP) y de seguridad (Seguridad-MAGERIT), se pueden considerar como resultado de un proceso *ad hoc* de análisis de sus respectivos dominios: ambos perfiles de requisitos se crearon sin seguir un método definido para el desarrollo para reutilización. Esto fue posible porque ambos dominios estaban muy estructurados, al considerar como fuente de los requisitos documentos legislativos, la LOPD y el RMS, o métodos bien definidos de análisis y gestión de riesgos como MAGERIT. Por este motivo fue relativamente directo estudiarlos y estructurar el conocimiento en catálogos de requisitos. Pero ¿qué ocurre cuando el dominio del problema es amplio y el conocimiento no está previamente estructurado en un conjunto de documentos oficiales? Pensamos que el ingeniero de requisitos necesita asistencia metodológica en el proceso de análisis del dominio, y que la aproximación *ad hoc* basada en lenguaje natural no es suficiente para obtener un catálogo de requisitos de calidad. En la definición actual de SIREN se ha añadido una tarea T1 de corte general para abordar este problema. En el siguiente capítulo ofrecemos una extensión de SIREN para abordar el análisis de un dominio vertical complejo como el de los sistemas teleoperados para mantenimiento de cascos de buques, extensión que pensamos se puede utilizar en el ámbito de otros sistemas teleoperados.

En el ámbito del DGS, se ha presentado un catálogo de buenas prácticas (riesgos y salvaguardas asociadas) que compila el estado del arte en este campo, y que con un soporte automatizado adecuado, como por ejemplo una wiki, podría proporcionar una fuente común (1) donde los profesionales del desarrollo de software podrían comprobar el estado de la investigación en el campo del DGS y podrían conocer los riesgos que pueden encontrar en sus proyectos; (2) los investigadores podrían conocer rápidamente el estado de la investigación en este campo y qué áreas merecen más atención; y (3) cada grupo –desarrolladores, investigadores– podría actualizar el catálogo con sus propios resultados y utilizar este catálogo para formar una comunidad de estudio en este campo. Se debe tener en cuenta que se trata de una primera versión del catálogo de riesgos y salvaguardas, y por ello algunos riesgos todavía no tienen salvaguarda asociada, pues no se ha encontrado en la literatura y no se ha propuesto ninguna. El método SIRENgsd, que extiende SIREN para considerar los riesgos y salvaguardas del catálogo y las cuestiones inherentes al DGS, constituye una primera propuesta para considerar la reutilización de requisitos en el DGS, que todavía no ha sido validada en un caso de estudio real.

3 SIRENspl: Una evolución de SIREN para líneas de productos

3.1 Introducción

La idea de reutilizar software es casi tan antigua como el desarrollo del mismo. Desde el inicio de la visión del *software como producto*, en los años sesenta, generalmente se ha conseguido reutilizar software a través de propuestas *ad hoc*, a pequeña escala, centradas en el código y que no han tenido un amplio impacto en la industria. Con el desarrollo de software basado en componentes, que se comienza a generalizar en la segunda mitad de los noventa, se consigue una mejora apreciable en la reutilización en el nivel del código. Sin embargo, numerosos autores abogan por una reutilización de grano más grueso, que requiere disponer de elementos software reutilizables o *assets* en todos los niveles del desarrollo. Como se ha indicado en la Sección 1.1.2, Rine y Nada [262] han mostrado empíricamente que el nivel de reutilización determina la efectividad de las mejoras en productividad, calidad y tiempo de desarrollo, concluyendo que se obtienen mayores beneficios cuando la reutilización se considera durante las primeras etapas del ciclo de vida de desarrollo de software.

Como recogen van der Linden *et al.* [288], Parnas [241] es el primero en proponer el concepto de *familias de productos* en los años setenta. Posteriormente, en los años ochenta, Neighbors [220] introduce la idea de enfocar en un dominio específico como base para el desarrollo de assets reutilizables, con un trabajo que es precursor de los lenguajes específicos de dominio. El concepto de línea de productos se define completamente en los primeros años noventa, con trabajos como los del método FODA (*Feature-Oriented Domain Analysis*) [160]. En aquellos años un conjunto de grandes empresas comienza a invertir con fuerza en el campo de las líneas de productos, de manera que a partir de la segunda mitad de los años noventa el concepto de línea de productos va cobrando protagonismo en la ingeniería del software y se forma una comunidad de investigación y de práctica en líneas de productos.

Con las líneas de productos software se consigue reutilización a gran escala en un determinado dominio, consiguiendo mejorar costes, tiempo de desarrollo (*time-to-market*) y calidad del producto final [288]. Se habla de "reutilización a gran escala" pues se produce la reutilización de hasta el 90% del software de un producto concreto dentro de la línea. La reutilización da lugar a una mejora en el coste del software de órdenes de magnitud en relación con el desarrollo de dicho software: costes y tiempo de desarrollo se reducen drásticamente mediante el enfoque de líneas de productos. Con este enfoque también se reducen los riesgos del proyecto, pues los proyectos son más acotados y su tamaño es más pequeño. Además se mejora la calidad del producto final, pues las nuevas aplicaciones consisten en un amplio porcentaje de componentes maduros y comprobados. Esta mejora de la calidad se traduce en sistemas más fiables y seguros. Para conseguir estos beneficios, la ingeniería de líneas de productos requiere cambios profundos en la organización, así como invertir en la construcción de assets reutilizables (van der Linden *et al.* [288] señalan que se suele conseguir la amortización de la inversión en el dominio con unos tres productos por término medio).

Los sistemas teleoperados para mantenimiento de cascos de buques (en adelante, simplemente STO) son sistemas robóticos extremadamente útiles en tareas tales como limpiar o pintar el casco de un barco [101, 240]. Estos sistemas constituyen una línea o familia de productos (ver Sección 1.1.3), pues son un conjunto de sistemas intensivos en

software que comparten un conjunto de características comunes que satisfacen las necesidades específicas de un segmento del mercado [57]. En los últimos años se han desarrollado arquitecturas software de referencia para los STO (por ejemplo, véase Álvarez *et al.* [25] o Iborra *et al.* [136]). Estas arquitecturas genéricas están formadas por un conjunto común de artefactos software reutilizables (*assets*) a partir de los cuales se acelera el desarrollo de los STO, que suelen compartir un alto número de comandos de una implementación a otra. Por comandos entendemos básicamente operaciones que cambian el estado del sistema, como por ejemplo operaciones de movimiento de una articulación del robot o de cambio del modo de funcionamiento del sistema. El esfuerzo dedicado a la reutilización de requisitos en los STO ha sido mucho menor que el dedicado a la reutilización de componentes arquitectónicos. En la práctica, en este tipo de sistemas robóticos, las especificaciones de requisitos no han sido desarrolladas rigurosamente con el objetivo de promover la reutilización.

En este contexto, nos hemos planteado la conveniencia de iniciar el desarrollo de nuevos productos dentro de esta línea de productos a partir de un nivel de abstracción más alto –desde los requisitos de la línea de productos en lugar de desde la arquitectura genérica de la misma–, con la obtención de obtener lo que se puede denominar intuitivamente un *modelo del dominio* de los STO (en la Sección 3.4.2 se discute de forma más rigurosa el significado de este término y de otros relacionados). En definitiva, la idea consiste en dotar a los STO de una interfaz que permita un razonamiento más abstracto sobre los productos de la línea.

Ante un dominio vertical complejo como el de los STO, y a la luz de las conclusiones de la Sección 2.11 sobre SIREN, nos podemos plantear si el método SIREN proporciona las técnicas más adecuadas para realizar un análisis del dominio de los STO: (1) por un lado, según la experiencia del investigador en los dominios de protección de datos personales [285] y de seguridad [284], resumida en el Capítulo 2, un riesgo en la elaboración mediante SIREN de un catálogo de requisitos reutilizables para un dominio amplio y complejo como el de los STO es que se puede obtener un catálogo preciso y correcto pero difícil de manejar, al estar formado por largas *listas de la compra* de requisitos textuales, relacionados mediante trazas; y (2) por otro lado, sabemos que existen técnicas gráficas más potentes para capturar la *variabilidad* de una línea de productos (básicamente las cuestiones comunes y variables de los STO) que los requisitos textuales en lenguaje natural (por ejemplo [47, 249, 269, 271]). En conclusión, hemos decidido extender el método SIREN presentado en el Capítulo 2 con nuevas técnicas que permiten capturar de forma más expresiva la variabilidad de los requisitos de la línea de productos. Se inicia de esta manera una nueva línea de investigación en el marco de esta tesis doctoral, Ingeniería de Requisitos (IR) para líneas de productos, a partir de la línea de investigación base de IR y reutilización, descrita en el Capítulo 2. Así pues, el modelo de procesos, las técnicas y las guías de SIREN se extienden para dar soporte al análisis de este nuevo dominio, dando lugar a la propuesta que denominamos *SIRENspl* (donde *spl* procede de *software product line*). Esta propuesta está soportada por el prototipo de una herramienta CARE desarrollada con Eclipse EMF/GMF.

En este capítulo se presenta un caso de estudio en el que se lleva a cabo un análisis del dominio de los STO. Este caso de estudio resulta en un modelo del dominio de los STO junto con unas lecciones aprendidas. El modelo del dominio de los STO sirve para (1) acelerar la comprensión de la línea de productos (indicando sus elementos comunes y

variables); (2) para obtener en contexto los requisitos reutilizables de la línea de productos; (3) para tomar decisiones de forma eficiente sobre la especificación de los nuevos sistemas desarrollados dentro de la línea de productos; y (4) para reutilizar la arquitectura de referencia y los comandos utilizados en la implementación de la línea de productos.

Para mejorar la aplicabilidad de la propuesta y la comprensión de sus resultados, en el desarrollo de SIRENspl a través del caso de estudio se han elegido técnicas de análisis del dominio bien conocidas. Con relación a este punto Kaindl *et al.* [158] afirman que es interesante "una aplicación más amplia de las aproximaciones existentes en IR", y que "la IR será inmadura hasta que construyamos sobre lo que otros han hecho, en lugar de inventar otra nueva técnica de modelado". Las técnicas seleccionadas para SIRENspl han sido ordenadas en un marco conceptual sencillo y han sido extendidas para dar cuenta de las particularidades del dominio. Glass *et al.* [117] señalan que tradicionalmente las soluciones para dominios específicos han recibido poca atención en la literatura de ingeniería del software, que se ha centrado en soluciones genéricas. Este estudio, por el contrario, presenta una aproximación para el modelado de un dominio específico, el de los STO.

El desarrollo de nuestra propuesta para el análisis de los STO está íntimamente ligado al caso de estudio, desarrollado mediante investigación en acción (I-A, ver Sección 1.3.2), un método de investigación en el cual los investigadores trabajan con los expertos en el dominio para generar una comprensión mayor del fenómeno determinado, ofreciendo soluciones prácticas a problemas concretos. Así pues, la estructura de este capítulo refleja esta relación tan cercana entre la investigación y los STO, el objeto investigado en dicho caso de estudio. Con la presentación de la propuesta se está presentando también la resolución del caso de estudio. Este capítulo se estructura de la siguiente manera: en la Sección 3.2 se describe brevemente el dominio de los STO. En la Sección 3.3 se hace un breve repaso del estado del arte en IR para líneas de productos. En la Sección 3.4 se describe el caso de estudio, incluyendo (1) el diseño del proceso de la investigación; (2) el marco establecido para el análisis de este dominio, en el que se ubica el modelo de características (que enriquece el de FORM, un método bien conocido en desarrollo de líneas de productos [161]), el modelo de casos de uso genéricos, el esquema conceptual del dominio, las plantillas de atributos de calidad y la descripción de comandos; (3) el modelo de procesos en el que se ubicarían las técnicas anteriores; (4) las lecciones aprendidas en esta experiencia; (5) la validez de los resultados obtenidos; y (6) los trabajos relacionados. En la Sección 3.5 se presenta la herramienta CARE de soporte al modelado con SIRENspl. Finalmente, en la Sección 3.6 se presentan las conclusiones.

3.2 Sistemas teleoperados para mantenimiento de cascos de buques

Un problema crítico para la industria naval europea es la preparación de las superficies del casco para su pintado de manera respetuosa con el medio ambiente. La limpieza periódica del casco del buque para decaparlo antes de su repintado (*hull blasting*) es una

operación crítica en el mantenimiento de buques para la cual existen hasta la fecha algunas soluciones parciales, como turbinas de chorreado mediante granalla para superficies verticales (*grit blasting turbines for vertical surfaces*) o unidades para decapar mediante chorreado de agua a presión (*water blasting units*). Las soluciones basadas en granalla se restringen usualmente al chorreado completo (*full blasting*) en superficies verticales. Las soluciones basadas en agua a presión son caras y no han mostrado el mismo rendimiento y calidad en la preparación de la superficie como los sistemas de chorreado basado en granalla. En un futuro, se prevé la ampliación de la línea de productos a la realización de otras actividades de mantenimiento de barcos, como el pintado de la superficie del casco o ciertas clases de soldaduras. Es por esto que en el título de esta sección hablamos de sistemas teleoperados para mantenimiento de cascos de buques en lugar de sistemas teleoperados para limpieza de cascos de buques.

El objetivo global del proyecto europeo EFTCoR (*Environmental Friendly and Cost-Effective Technology for Coating Removal*, V Programa Marco, [93]) ha sido el desarrollo de una nueva tecnología de teleoperación para la limpieza de cascos de buques. En el ámbito de este proyecto, la División de Sistemas e Ingeniería Electrónica (DSIE) de la Universidad Politécnica de Cartagena (UPCT) ha desarrollado una familia de robots (Figura 15) para la limpieza del casco mediante chorreado basado en granalla (*hull grit blasting*), que ha sido capaz de obtener una preparación de la superficie de alta calidad al tiempo que ha reducido drásticamente los residuos generados y ha producido cero emisiones al medio ambiente. Además de la DSIE, que dispone de una amplia experiencia en el desarrollo de este tipo de robots de servicios [136], en el proyecto EFTCoR ha participado un grupo de investigación de la Universidad Politécnica de Madrid y ocho empresas europeas del sector.

En la Figura 16 se muestra un diagrama de bloques de los subsistemas considerados en el desarrollo de los robots EFTCoR y sus relaciones. Este diagrama es una descripción conceptual que sirve para presentar brevemente las funcionalidades de dichos subsistemas, que son los siguientes:

- El *Subsistema de Monitorización*, que abarca la funcionalidad que concierne las necesidades de información y gestión relacionadas con el mantenimiento del barco.
- El *Subsistema de Visión*, que recoge la funcionalidad relacionada con la inspección del casco y determina las áreas del mismo que deben ser tratadas y su estado antes y después de la preparación de la superficie. Esta información constituye una de las entradas al Subsistema de Monitorización.
- La *Unidad de Control de Dispositivos Robóticos (RDCU, Robotic Device Control Unit)*, que está a cargo de controlar los dispositivos robóticos (sistemas de posicionamiento y herramientas) utilizados en las tareas de mantenimiento de acuerdo con las órdenes y eventos generados por otros subsistemas.
- El *Subsistema de Reciclado*, que está encargado de recuperar los residuos desde las áreas de trabajo y reciclarlos. Debido a que tales residuos han de ser recuperados en línea, existe una fuerte relación entre la operación de la cabeza limpiadora (*cleaning head, blasting tool*) y la operación del Subistema de Reciclado.

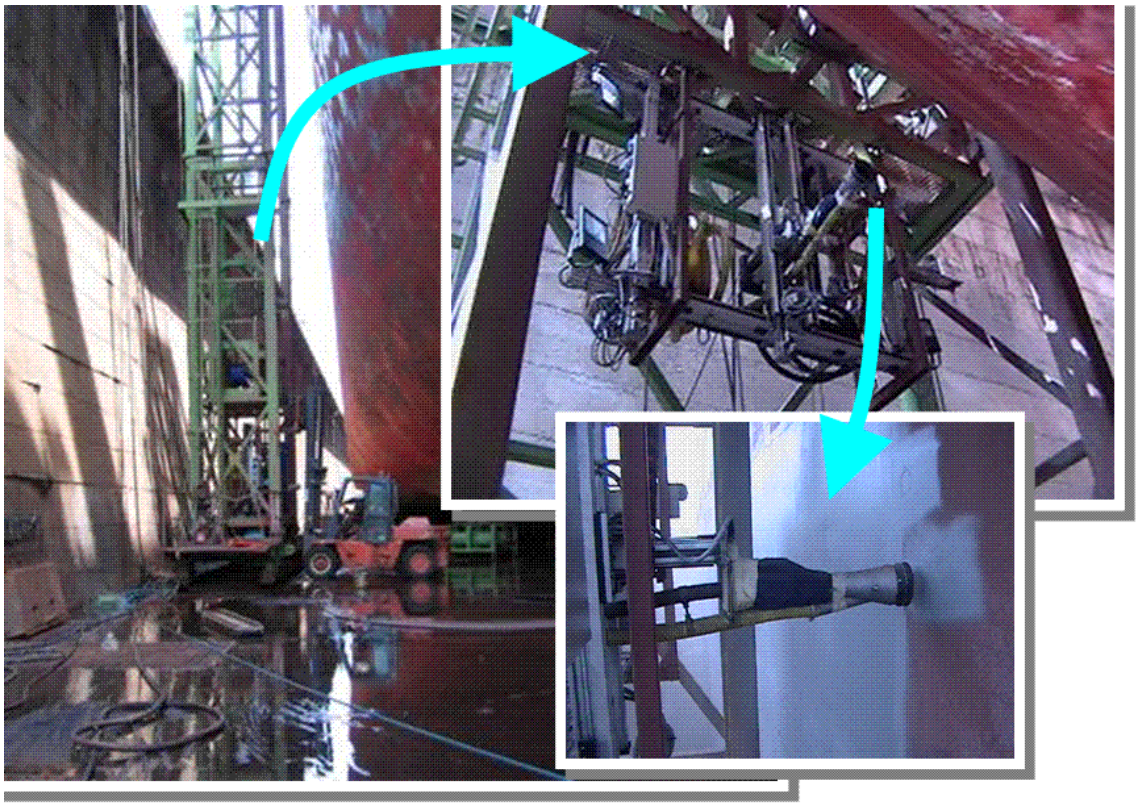

Figura 15 Un sistema teleoperado para limpieza de cascos de buques (*fuente: DSIE*)

Los dispositivos robóticos consisten básicamente de una cabeza limpiadora y de unos dispositivos de posicionamiento (*positioning devices*). El posicionamiento de las cabezas limpiadoras sobre el casco del buque es un problema que se puede resolver de varias maneras. En el desarrollo del proyecto EFTCoR se han propuesto las siguientes soluciones:

- Se ha desarrollado una familia de sistemas especializados de bajo coste en lugar de un único sistema de propósito general.
- La naturaleza distinta de las dos operaciones básicas de limpieza, limpieza completa (*full blasting*) y limpieza de manchas (*spot blasting, spotting*), conduce a sistemas diferentes.
- El sistema de posicionamiento global ha sido dividido en dos subsistemas de posicionamiento independiente: (1) un sistema de posicionamiento *primario*, que es capaz de posicionar cargas pesadas sobre grandes superficies (el casco del buque completo); y (2) un sistema de posicionamiento *secundario*, que puede ser montado sobre el primario, capaz de posicionar una cabeza limpiadora ligera con la precisión requerida para limpiar manchas sobre pequeñas superficies (de 4 a 10 m^2).
- Los sistemas de posicionamiento primario y secundario han sido diseñados de tal manera que son posibles distintas combinaciones de sistemas de posicionamiento primario y secundario.

- Un vehículo con una cabeza limpiadora, *Lázaro*, ha sido desarrollado para llegar a áreas que no son alcanzables con una combinación razonable de sistemas primario y secundario.
- Siempre que ha sido posible se ha adoptado o adaptado una solución comercial para hacer el trabajo requerido.

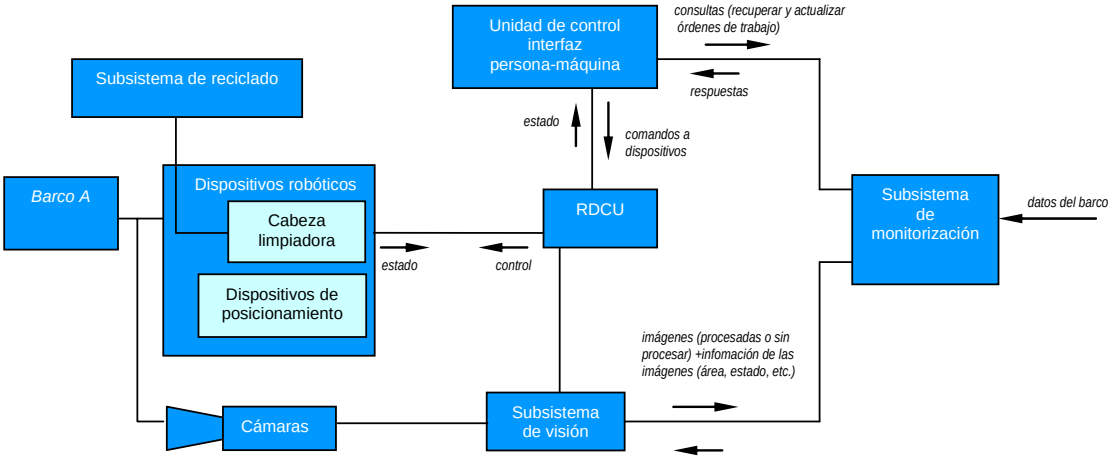

Figura 16 Un diagrama de bloques que muestra los subsistemas de los STO (*fuente: DSIE*)

3.3 Ingeniería de requisitos para líneas de productos

En la Sección 1.1.3 se han introducido brevemente conceptos básicos como línea de productos software, IR para líneas de productos, ingeniería del dominio e ingeniería de la aplicación, IR del dominio (o análisis del dominio) e IR del producto (o IR de la aplicación). En esta sección seguimos a Nicolás y Toval [227, 228] para describir cuestiones generales relacionadas con la IR para líneas de productos, que giran alrededor del concepto de variabilidad y que se ejemplifican con la línea de productos de los STO.

Los requisitos de una línea de productos definen los productos de dicha línea y sus características comunes y variables. Según el texto clásico de Clements y Northrop [57] sobre líneas de productos, la IR para líneas de productos debe gestionar los requisitos de la línea de productos y los requisitos de los productos concretos de la línea. Dentro de la IR para líneas de productos se debe distinguir, por tanto, un proceso de IR del dominio y un proceso de IR de la aplicación. La IR de la aplicación debe incorporar un mecanismo mediante el cual el conjunto de requisitos para un producto concreto sea producido de manera fácil y rápida a partir de los requisitos de la línea de productos.

La IR para líneas de productos suele involucrar a expertos del dominio y toma como fuentes de información los modelos del dominio existentes y el *alcance* de la línea de productos. El alcance de la línea de productos limita o define los productos incluidos en

ella, mientras que los requisitos de la línea de productos refinan dicho alcance definiendo de forma más precisa las características de los productos en la línea de productos. Ambos, alcance y requisitos, están muy relacionados y evolucionan de manera conjunta.

Clements y Northrop [57] han estudiado los riesgos principales que afectan la IR para líneas de productos, y determinan que son los siguientes:

- No se distingue bien entre los requisitos comunes a toda la línea y los requisitos específicos de un producto.
- Se especifican requisitos con una generalidad insuficiente o excesiva.
- Se determinan puntos de variación incorrectos.
- No se contemplan requisitos de calidad como rendimiento, fiabilidad o seguridad. La variabilidad de la línea de productos puede afectar a tanto a requisitos funcionales como de calidad.

Estos autores afirman que los riesgos que afectan el análisis del dominio pueden también perjudicar la IR de la línea de productos:

- El proceso de análisis del dominio no se comprende bien, de manera que se discuten cuestiones de diseño e implementación demasiado pronto, o bien se dedica excesivo tiempo al análisis, cayendo en la *parálisis del análisis* [298].
- La información del dominio no está bien documentada, se encuentra sólo en la mente de algunos expertos del dominio: como mínimo, deberían estar documentadas las decisiones sobre qué es común, qué es variable y qué está fuera de la línea de productos, así como las justificaciones de tales decisiones.
- No se tiene acceso a los expertos en el dominio.
- Existe falta de compromiso por parte de la dirección.

Finalmente, Clements y Northrop enumeran ciertas prácticas que pueden ser de especial utilidad en la IR de líneas de productos:

- Técnicas de análisis del dominio, entre las que destacan los modelos de características (ver Sección 3.4.3).
- El modelado de casos de uso y de *casos de cambio* (Sección 3.4.4).
- El modelado de los puntos de vista de los interesados (*stakeholders*), que facilita la identificación de conflictos y la adopción de compromisos (Sección 2.2.4.2).
- La traza de los requisitos hacia el resto de artefactos de desarrollo y hacia sus fuentes (Sección 2.2.3.4).

3.3.1 Ingeniería de requisitos del dominio y de la aplicación

En este apartado se describen brevemente los procesos de IR del dominio y de la aplicación siguiendo a Pohl *et al.* [249].

La IR del dominio engloba aquellas actividades que extraen, analizan, especifican y verifican los requisitos comunes y variables de la línea de productos:

a. La entrada de este proceso es el alcance de la línea de productos (que Pohl *et al.* denominan *hoja de ruta de productos*, *product roadmap*), que recoge las principales características comunes y variables de todas las aplicaciones de la línea, así como la programación de su introducción en el mercado. De cara a la gestión de los requisitos del dominio, la hoja de ruta de productos, por tanto, fija el alcance de la *plataforma* (*platform*, compuesta por los *assets* o artefactos reutilizables del dominio, almacenados en un repositorio), y describe los aspectos más importantes de la variabilidad de la línea de productos.
b. La salida de este proceso comprende los requisitos y el modelo de variabilidad de la línea de productos. La salida, por tanto, no incluye la especificación de requisitos para una aplicación determinada, sino los requisitos comunes y variables de todas las aplicaciones que se prevé desarrollar a partir de la línea de productos.

La IR del dominio se diferencia de la IR tradicional básicamente en los siguientes puntos:

- Los requisitos han de ser analizados para determinar cuáles son comunes a todas las aplicaciones y cuáles son específicos de una aplicación particular.
- Las posibles variantes de los requisitos son documentadas explícitamente.
- La IR del dominio trata de anticipar cambios en los requisitos a partir de leyes, estándares, cambios tecnológicos y necesidades del mercado.

La IR de la aplicación engloba aquellas actividades que sirven para desarrollar la especificación de los requisitos de una aplicación de la línea de productos:

a. Las entradas de este proceso son los requisitos del dominio y la hoja de ruta de productos, junto con las principales características de la aplicación correspondiente. Además, se pueden considerar requisitos específicos de la aplicación que no habían sido especificados durante la IR del dominio.
b. La salida de este proceso es la especificación de requisitos de una aplicación particular.

La IR de la aplicación se diferencia básicamente de la IR tradicional en los siguientes puntos:

- La extracción de requisitos está basada en los requisitos comunes y variables de la línea de productos. La mayor parte de los requisitos no se extraen por primera vez, sino que son derivados de los requisitos del dominio existentes.
- Durante la extracción, las diferencias (*deltas*) entre los requisitos de la aplicación y los requisitos del dominio (las capacidades disponibles en la plataforma) han de ser detectadas, evaluadas (en relación con el esfuerzo de adaptación) y documentadas. Si el esfuerzo de adaptación necesario se conoce pronto, si fuera preciso se podrían tomar compromisos para reducirlo e incrementar la cantidad de reutilización de la plataforma.

3.3.2 Variabilidad en la especificación requisitos

Una línea de productos software tiene por objetivo dar soporte a un abanico de distintos productos que satisfacen bien a distintos clientes individuales o bien a distintos segmentos del mercado. La variabilidad es por tanto un concepto clave en cualquier enfoque de línea de productos. No se trata de comprender cada sistema individual de forma completamente independiente, sino de considerar la línea de productos como un todo y especificar las variaciones de los distintos sistemas individuales. Como afirman van der Linden *et al.* [288], la variabilidad debe ser definida, representada, explotada, implementada, evolucionada, etc. –en una palabra, *gestionada*– a través del proceso completo de ingeniería de la línea de productos. La gestión de la variabilidad en los requisitos del dominio y de la aplicación es crucial en una línea de productos (y por ende en una factoría de software).

Así pues una cuestión esencial cuando se modelan los requisitos de una línea de productos es el modelado de la variabilidad. Como afirman Pohl *et al.* [249], en un enfoque de línea de productos el software contiene *puntos de variación* que especifican *variantes* –opciones del sistema que se dejan abiertas durante la ingeniería del dominio (*core asset development*)–. En algún momento del proceso de ingeniería de la aplicación (*product development*) se han de instanciar los puntos de variación de manera que el comportamiento del producto final quede completamente especificado. Se instancian sólo aquellos puntos de variación que son relevantes para la aplicación en cuestión; el resto se puede dejar sin instanciar.

Una clasificación sencilla de la variabilidad en una línea de productos es la de van der Linden *et al.* [288], que distinguen tres tipos fundamentales de variabilidad:

1. *Común* (*commonality*): una característica (funcional o no funcional) puede ser común a todos los productos de la línea de productos. Se implementa como parte de la plataforma.
2. *Variable* (*variability*): una característica puede ser común a ciertos productos, pero no a todos. Por tanto se debe modelar como una posible variabilidad y debe ser implementada de manera que permita que sólo unos productos determinados de la línea soporten dicha característica.
3. *Específica de producto* (*product-specific*): una característica puede ser parte de un único producto (al menos en un futuro predecible). A menudo tales características no son requeridas por el mercado, sino por clientes individuales. Estas características no están integradas en la plataforma, pero la plataforma debe ser capaz de soportarlas.

En esta sección se muestran técnicas para representar la variabilidad en una especificación de requisitos. Al hilo de lo presentado en la Sección 2.2.4.3, la variabilidad se puede considerar un *aspecto* más en el desarrollo de software. En esta sección se presenta un breve resumen del estado de la práctica en la gestión de la variabilidad en los requisitos, a través de dos estrategias: (1) el uso de modelos de requisitos enriquecidos con construcciones para modelar la variabilidad; y (2) el uso de un *modelo de variabilidad independiente* relacionado a través de trazas con los artefactos de requisitos correspondientes. Pohl *et al.* [249] describen en profundidad estos dos enfoques:

- El primer enfoque consiste en usar modelos de requisitos estándar en los cuales está inmersa la especificación de la variabilidad. Se extienden modelos de requisitos, como modelos de características y modelos de casos de uso, que se enriquecen con mecanismos para plasmar la variabilidad. En esta línea, la Sección 3.3.2.1 describe los modelos de características y la Sección 3.3.2.2 los modelos de casos de uso.
- El segundo enfoque es más reciente. Consiste en definir un *modelo de variabilidad* independiente de los modelos de requisitos y mantener relaciones de traza entre este modelo de variabilidad y los modelos de requisitos. La idea básica es que el modelo de variabilidad incluye los puntos de variación y las alternativas de la línea de productos y es independiente del resto de modelos del desarrollo: las alternativas en el modelo de variabilidad se trazan a subconjuntos de los modelos de desarrollo. Por ejemplo, una alternativa se puede trazar a ciertas características y casos de uso en el nivel de los requisitos. Con este enfoque basado en un modelo de variabilidad independiente –que se describe en detalle en la Sección 3.3.2.3 y se ejemplifica en las secciones 3.3.2.4 y 3.3.2.5 con el *modelo de variabilidad ortogonal* de Pohl *et al.* [249]– se trata de resolver ciertos problemas asociados a la primera vía.

3.3.2.1 Modelos de características

Un modelo de características (*feature model*) –véase FODA (*Feature-Oriented Domain Analysis*) [160], FORM (*Feature-Oriented Reuse Method*) [161] o PLA (*Product Line Analysis*) [68]– sirve para especificar de forma intuitiva la visión que los interesados tienen de la línea de productos, en particular de sus requisitos funcionales y no funcionales o de calidad. El término *característica* (*feature*) es un término sobrecargado, pues ha sido usado en muchas ocasiones en IR con significados parecidos. En análisis del dominio las características fueron introducidas por primera vez por Kang *et al.* en FODA (*Feature-Oriented Domain Analysis*) [160]. Para FODA, las características son abstracciones de las capacidades del dominio que deben ser implementadas, probadas, entregadas y mantenidas. En el modelo de características se especifican las capacidades (incluyendo los requisitos funcionales y no funcionales) y las restricciones tecnológicas que deben y que pueden aparecer en los productos de la línea de productos (*commonality* y *variability*), y que son visibles al usuario final. Dado que una característica se puede descomponer en varias *subcaracterísticas* (obligatorias, opcionales o alternativas), el modelado de características produce una descomposición jerárquica en forma de árbol o de un conjunto de árboles. Por ejemplo, en la Figura 17 se muestra un extracto del diagrama de características para los STO.

La notación utilizada en la Figura 17 es la típica de un modelo de características, como la definida por Kang *et al.* en FODA (*Feature-Oriented Domain Analysis*) [160] o en FORM (*Feature-Oriented Reuse Method*) [161], un método que extiende FODA al proceso de diseño arquitectónico y de componentes. Con esta notación la descomposición AND de una característica se muestra mediante las relaciones directas entre nodos en la figura, mientras que la descomposición OR se muestra mediante un arco que cruza las relaciones que unen la característica con sus distintas alternativas. Las características opcionales en una descomposición AND son etiquetadas a su vez con un círculo blanco. Cada característica se describe complementariamente a través de una plantilla, como por ejemplo la propuesta por Kang *et al.* [161].

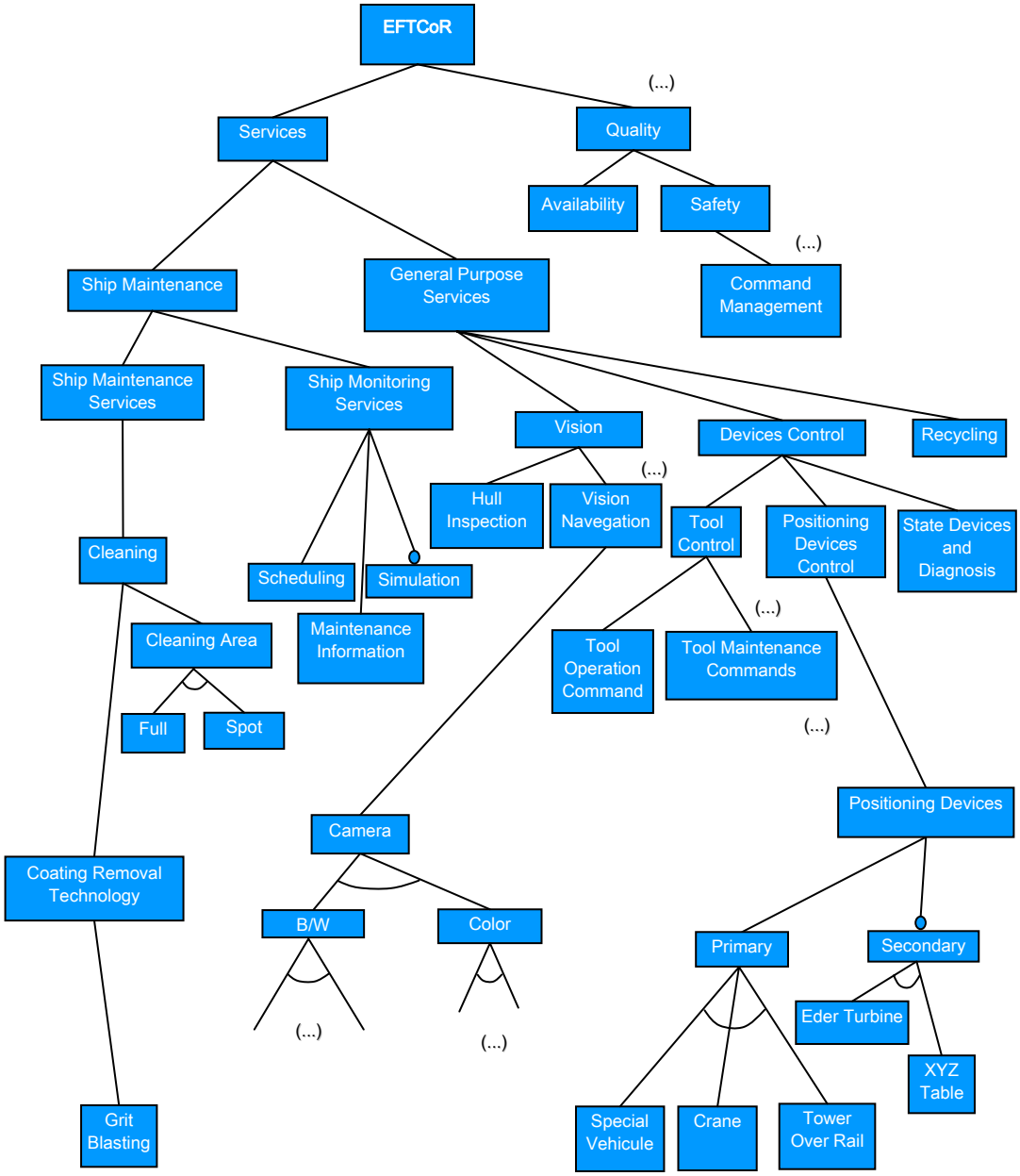

Figura 17 Diagrama de características de los STO con la notación de FODA/FORM (extracto)

Lee *et al.* [182] afirman que desde su aparición el modelo de características ha sido usado extensamente en IR para líneas de productos en comparación con otras técnicas de análisis del dominio, lo cual atribuyen básicamente a tres razones: (1) los clientes suelen expresar de forma intuitiva sus necesidades en términos de las "características que el nuevo sistema ha de tener o proporcionar", entendidas como abstracciones de las capacidades del sistema que deben ser implementadas, probadas, entregadas y mantenidas; (2) el modelo de características es una forma efectiva de capturar las partes comunes y variables de los productos de un dominio: clientes y desarrolladores pueden comprender de forma intuitiva un modelo de características, con el cual se puede explorar rápidamente el dominio para conocer las cuestiones principales y determinar puntos comunes y variables en la línea de productos: durante la IR del producto, se puede navegar a través del *espacio de decisión* que viene dado por las características,

seleccionando unas u otras; y (3) el modelo de características juega un papel central para desarrollar, parametrizar y configurar los artefactos software reutilizables (*assets*) del dominio, como el modelo de requisitos del dominio, el modelo arquitectónico de referencia y los componentes de código reutilizable.

Existe un cierto solapamiento en la definición de características (*features*) y metas (*goals*, ver Sección 2.2.4.1). Las primeras son características de un sistema visibles por un usuario final, mientras que las segundas son objetivos que el sistema debe conseguir. Las metas han sido introducidas por la comunidad de IR para expresar las intenciones de alto nivel que conciernen a un sistema, y que después son refinadas en requisitos. Las características, en cambio, han sido introducidas por la comunidad de diseño de software para abstraer un diseño arquitectónico de alto nivel, es decir, para expresar los requisitos de alto nivel de la arquitectura. Distintos autores han ligado ambos conceptos (véase, por ejemplo, la propuesta de Baixauli *et al.* [124], que integra características, metas y casos de uso).

El desarrollo de software se facilitaría si el software se pudiera desarrollar a partir de la identificación de sus características, esto es, si se articularan mecanismos para que las características se representaran explícitamente en el diseño y en el código y si fuera posible generar una aplicación únicamente componiendo las características deseadas. Esta es la idea esencial que subyace en un paradigma de desarrollo de software denominado *Feature-Oriented Software Development* (FOSD), un paradigma de síntesis de programas en líneas de productos software. FOSD estudia el análisis, diseño, e implementación de características en líneas de productos software, incluyendo el estudio de la modularidad de las características, las técnicas de análisis y diseño para soportar la síntesis de programas basada en características y las herramientas de soporte. Actualmente existen numerosos métodos, lenguajes y herramientas para FOSD (ver, por ejemplo, la compilación realizada en [107]).

3.3.2.2 Modelos de casos de uso

El modelo de características proporciona mecanismos para plasmar la variabilidad de una línea de productos, pero se puede argumentar que la variabilidad no se circunscribe únicamente al modelo de características. Por ejemplo, puede existir variabilidad en los escenarios de uso de una misma operación soportada por productos distintos de la línea de productos. El modelo de casos de uso describe las interacciones entre los actores y el sistema que se producen durante la ejecución de ciertas características que capturan una funcionalidad ejecutable de la línea de productos, y sería por tanto el lugar adecuado para plasmar la variabilidad anterior.

Existe un consenso –ver por ejemplo [122]– en cuestionar la idoneidad de los casos de uso *clásicos* para el modelado de líneas de productos, introduciendo de una u otra manera el concepto de variabilidad en los casos de uso, que a veces son referidos como casos de uso *genéricos*. Se han realizado distintas propuestas para contemplar la variabilidad de una línea de productos en los casos de uso (por ejemplo [96, 123, 129, 155, 289]). En un caso de uso genérico, algunos pasos pueden variar dependiendo del producto concreto de la familia de productos o de ciertos parámetros. Se puede establecer una relación N:M entre características y casos de uso. La traza se puede establecer –en cualquier sentido– entre casos de uso y características o bien, de modo más fino, entre pasos dentro del caso de uso y características.

Siguiendo a Gomaa [122], si se dispone de un modelo de características como interfaz de alto nivel de la línea de productos, el modelo de casos de uso se puede estructurar por características, expresando con estas los puntos de variación de la línea de productos. Para hacer más legible el diagrama de casos de uso, Gomaa [122] propone el etiquetado de los casos de uso opcionales y alternativos con los estereotipos «optional» y «alternative» (ver Figura 18). De esa forma se aprecia visualmente mejor qué casos de uso son opcionales y alternativos, aunque para encontrar los detalles de la variabilidad se debe acudir al modelo de características. En la plantilla de descripción del caso de uso hay que modelar la variabilidad intrínseca a los casos de uso, esto es, la que se refiere a las posibles variaciones en los pasos de los escenarios de interacción, según se seleccionen unas características u otras.

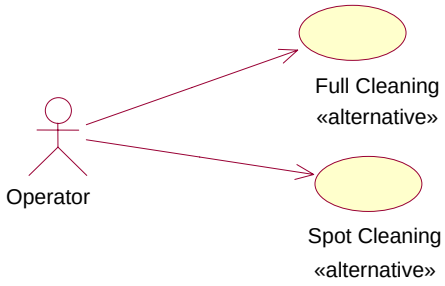

Figura 18 Casos de uso relacionados con la limpieza del casco (extracto)

3.3.2.3 Modelos de variabilidad generales

Aunque el uso de modelos de características para representar la variabilidad está muy extendido [288], una tendencia reciente consiste en la definición de modelos de variabilidad que pueden ser usados para definir la variabilidad en todos los niveles del desarrollo. Según van der Linden *et al.* [288], existe un cierto consenso que considera que estos modelos de variabilidad son más sencillos de aplicar en proyectos complejos y que son más escalables. Un modelo de variabilidad define los puntos de variación de la línea de productos, las variantes ofrecidas dentro de la línea de productos para cada punto de variación, y las dependencias y restricciones que se deben considerar cuando se derivan aplicaciones dentro de la línea de productos.

La especificación explícita de la variabilidad en un modelo de variabilidad general mejora la comunicación de la variabilidad de una línea de productos, al proporcionar una abstracción de alto nivel de los artefactos variables, y mejora la toma de decisiones, obligando a documentar por qué se introduce un punto de variación o una variante.

En la medida de nuestro conocimiento, dentro de esta tendencia que utiliza modelos de variabilidad generales no se ha definido todavía un modelo de variabilidad estándar. En los siguientes trabajos se plantean modelos de variabilidad relevantes dentro del estado del arte:

- En su repaso del estado de la investigación en líneas de productos, Käkölä y Dueñas [159] incluyen dos modelos de variabilidad: (a) el *modelo de variabilidad ortogonal* de Pohl *et al.* [249]; y (b) el *modelo de variabilidad*

consolidado de Bayer *et al.* [37], que aglutina diversos enfoques anteriores con el objetivo de servir como punto de partida para la estandarización. El modelo de variabilidad ortogonal proporciona un metamodelo de variabilidad que sirve para definir la variabilidad en todos los niveles del desarrollo. El modelo de variabilidad ortogonal se relaciona con los artefactos de cada nivel del desarrollo mediante trazas. El modelo de variabilidad consolidado, en cambio, consiste en un metamodelo de variabilidad que se asocia con elementos de modelado, denominados *anotaciones*, que se incluyen directamente en cada modelo de la línea de productos. La variabilidad se resuelve mediante *resoluciones* a través de un *modelo de resolución*, que recoge las elecciones realizadas para instanciar el modelo de variabilidad en un producto concreto. En palabras de Bühne *et al.* [47], los modelos de variabilidad ortogonal y consolidado son dos modelos desarrollados en paralelo, desde diferentes perspectivas, si bien los resultados finales son muy parecidos. El modelo de variabilidad consolidado se ha definido desde una visión de diseño de la arquitectura, mientras que el modelo de variabilidad ortogonal se ha desarrollado desde una visión de IR. Este punto, junto con el hecho de que pensamos que el modelo de variabilidad ortogonal ha tenido un mayor impacto (por ejemplo, es el enfoque seleccionado por van der Linden *et al.* [288]), y que además nos parece más legible, nos ha hecho seleccionar éste para ser descrito con más profundidad en esta memoria de tesis doctoral (ver Sección 3.3.2.4).

- Sinnema *et al.* [271] enumeran los requisitos que debe satisfacer una notación para modelar la variabilidad en los distintos niveles del desarrollo, y definen un marco de gestión de la variabilidad que los satisface, denominado COVAMOF (*ConIPF Variability Modeling Framework*, donde *ConIPF* es el nombre del proyecto donde se desarrolló este marco). Los requisitos que Sinnema *et al.* imponen sobre un modelo de variabilidad son los siguientes: (1) los puntos de variación se deben representar uniformemente como entidades de primera clase en todos los niveles de abstracción (desde las características al código); (2) se debe permitir la organización jerárquica de la variabilidad; (3) se debe permitir la representación de dependencias (1:1 y N:M) como entidades de primera clase; y (4) se debe permitir el modelado de las relaciones entre dependencias. Creemos que la principal aportación de este marco tiene que ver con los requisitos (3) y (4), y es el tratamiento de las dependencias como entidades de primera clase, al igual que los puntos de variación. La *vista de variabilidad de COVAMOF (CVV, COVAMOF Variability View)* está formada así por dos vistas: la *vista de puntos de variación* y la *vista de dependencias*. Un lenguaje textual basado en XML, CVVL (*COVAMOF Variability View Language*) sirve para mostrar una representación textual de CVV.

- En el ámbito de PuLSE [254], uno de los métodos de desarrollo de líneas de productos más conocidos, Schmid y John [269] definen un marco para gestionar la variabilidad en todas las etapas del ciclo de vida, incluyendo un proceso para trasladar el marco general de gestión de la variabilidad a cualquier notación (por ejemplo, a una representación textual de los requisitos o a UML). A diferencia de los anteriores este marco está basado en el uso de *modelos de decisión*, que de forma tabular describen las decisiones (*variables de decisión*) que se deben adoptar para derivar un producto de la línea de productos. Por tanto en este enfoque los conceptos base de representación de la variabilidad son las decisiones que diferencian los distintos productos de la línea de productos. Los

requisitos que este enfoque impone sobre un modelo de variabilidad son los siguientes: (1) la definición de puntos de variación en los artefactos base del desarrollo; (2) la definición de elementos que puedan ser potencialmente ligados a estos puntos de variación; (3) la definición de las relaciones entre puntos de variación y elementos; (4) la definición de restricciones en las ligaduras entre puntos de variación y elementos; (5) un mecanismo de selección para definir los elementos que deben estar en un determinado producto; (6) el enfoque debe ser independiente de la notación; (7) el enfoque debe ser aplicable a todo tipo de artefactos durante el ciclo de vida; (8) el enfoque debe soportar la traza de las variaciones, tanto horizontalmente (en el mismo nivel de desarrollo) como verticalmente (entre distintos niveles de desarrollo); y (9) se debe poder estructurar el enfoque jerárquicamente, de forma que se mantenga escalable.

3.3.2.4 Modelo de variabilidad ortogonal

Pohl *et al.* [249] han identificado ciertos problemas cuando la variabilidad se especifica directamente en los modelos de desarrollo:

- Cuando la variabilidad se especifica en distintos modelos de desarrollo es muy difícil que toda la información se mantenga consistente.
- Es difícil determinar, por ejemplo, qué información sobre la variabilidad en los requisitos ha influenciado qué información en el diseño, la implementación o la prueba.
- Los modelos de desarrollo de software son de naturaleza compleja, y se pueden sobrecargar al incluir información sobre la variabilidad.
- Los conceptos afectados por la variabilidad difieren en los modelos de desarrollo, y se puede hacer difícil proporcionar una imagen global y consistente de la variabilidad.
- Se han identificado ambigüedades en la definición de la variabilidad con modelos usados tradicionalmente como los modelos de características (ver Sección 3.3.2.5).

En respuesta a estos problemas, el modelo de variabilidad ortogonal define la variabilidad de una línea de productos, proporcionando una visión de la variabilidad en todos los niveles del desarrollo a través de trazas con los modelos de análisis, de diseño, de componentes y de pruebas. En esta sección seguimos a Pohl *et al.* [249] para describir este modelo de variabilidad. Bühne *et al.* [47] también proporcionan una descripción adecuada del mismo.

En la Figura 19 se especifica en UML el metamodelo subyacente al modelo de variabilidad ortogonal, que se va a explicar a lo largo de esta sección. Cada elemento del modelo tiene un atributo –*anotación textual*– que permite, por ejemplo, registrar la motivación para introducir el elemento. Por simplicidad, los atributos no se muestran en la figura. Para representar la variabilidad de forma gráfica, a cada clase del metamodelo se asocia una notación gráfica como la mostrada en la Figura 20. En la Figura 21 se muestra un extracto del modelo de variabilidad ortogonal para el ejemplo de los STO recogido en la Figura 17.

Los elementos centrales del metamodelo son los conceptos de *punto de variación* y de *variante*. Nótese que una misma variante puede estar asociada con distintos puntos de variación (cardinalidad 1..n). Un punto de variación puede ser *interno* –tiene asociadas variantes que son visibles a los desarrolladores pero no a los clientes– o *externo* –tiene asociadas variantes visibles a desarrolladores y a clientes–. No es obligatorio que un punto de variación sea de consideración en todas las aplicaciones del dominio. Por ejemplo, en los STO, si el punto de variación *Área Limpieza (Cleaning Area)* toma el valor *Limpieza Completa (Full)*, no tiene sentido instanciar el punto de variación *Secundario (Secondary)*, pues basta con un primario para realizar la limpieza.

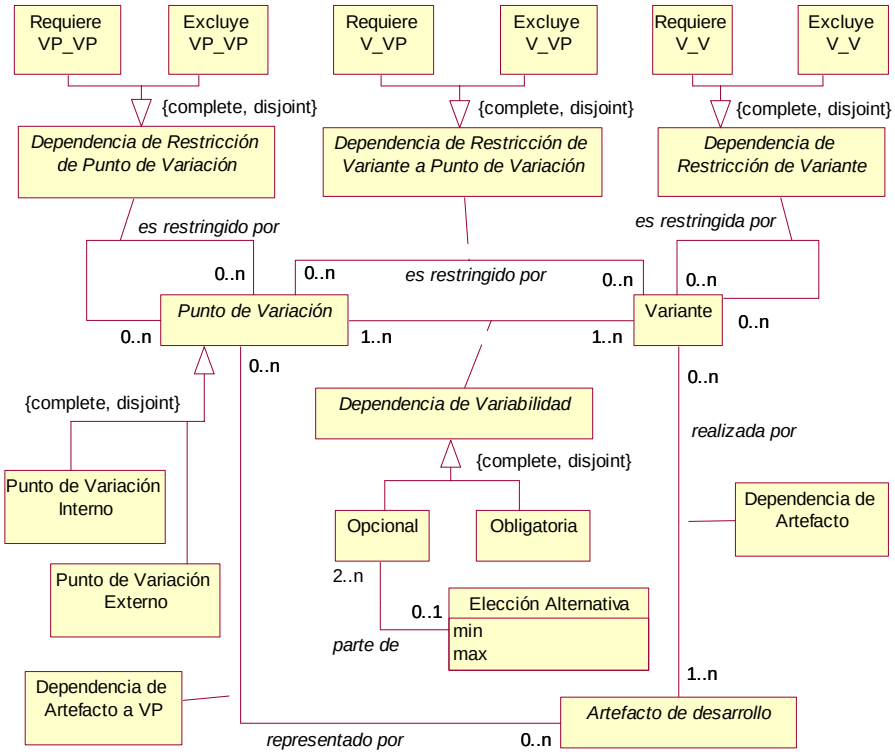

Figura 19 Metamodelo de variabilidad ortogonal (adoptado de Pohl *et al.* [249])

Puntos de variación y variantes se relacionan a través de *dependencias de variabilidad*, que pueden ser *opcionales* y *obligatorias*. Una dependencia opcional indica que la variante puede ser parte de una aplicación de la línea de productos, pero no tiene que serlo necesariamente. Una dependencia obligatoria indica que la variante debe ser seleccionada para una aplicación si y sólo si el punto de variación asociado es parte de la aplicación.

Una dependencia de variabilidad opcional puede formar parte de una *elección alternativa*, que agrupa un conjunto de variantes relacionadas a través de una dependencia de variabilidad opcional al mismo punto de variación y define el número mínimo y máximo de variantes opcionales que deben ser seleccionadas para este grupo. El rango por defecto es [1..1], que no se suele mostrar en los diagramas. Por ejemplo, en los STO las variantes Limpieza Completa (*Full*) y Limpieza Mancha (*Spot*, ambas relacionadas con Área Limpieza) forman una elección alternativa que puede instanciarse a Limpieza Completa, a Limpieza Mancha, o a ambas (por tanto, el rango es [1..2], ver

Figura 21), según se quiera un sistema que pueda realizar la limpieza de todo el casco, sólo de las zonas afectadas, o ambas posibilidades.

Existen *dependencias de restricción de variabilidad* que especifican relaciones *requiere* o *excluye* entre variantes, entre variantes y puntos de variación y entre puntos de variación y puntos de variación:

- Se puede establecer que una variante V1 *requiere* una variante V2 para funcionar correctamente. Consecuentemente, si V1 es seleccionada para una aplicación, V2 debe ser seleccionada. Por ejemplo, en los STO la variante *Limpieza Mancha* requiere que se disponga de las variantes *Primario* y *Secundario* (Figura 21). Análogamente, se puede especificar que una variante V1 *excluye* una variante V2.
- Se puede establecer que un punto de variación debe ser considerado (o no) en una aplicación si se selecciona una variante determinada en otro punto de variación, o incluso si se selecciona otro punto de variación. Por ejemplo, si se requiere una cámara en color y se selecciona el punto de variación *Color* no tiene sentido hacer lo propio con el punto de variación *Blanco y Negro (B/W)*.

La traza entre el modelo de variabilidad y cualquier artefacto del desarrollo afectado por la variabilidad se modela mediante *dependencias de artefacto*. Una variante puede ser trazada a artefactos de distintos niveles de granularidad (asociación *realizada por*, ver Figura 19), como por ejemplo a un caso de uso o a un paso de un escenario de un caso de uso. Un artefacto del desarrollo no tiene que estar relacionado necesariamente con una variante, pero una variante debe estar relacionada con al menos un artefacto del desarrollo. Por otro lado, hay situaciones en las que un artefacto del desarrollo debe representar un punto de variación. Por ejemplo, cuando en el diseño una clase abstracta implementa el comportamiento común de varias variantes. O cuando los desarrolladores quieren anticipar que hay variantes, que todavía no se han definido. Estas situaciones se solventan introduciendo una dependencia de artefacto entre un punto de variación y un artefacto del desarrollo (asociación *representado por*, ver Figura 19).

Cuando el modelo de variabilidad aumenta de tamaño y se hace complejo, una forma usual de gestionar la complejidad es la definición de *puntos de variación abstractos* que combinan puntos de variación concretos y predefinen las relaciones de dependencia de sus variantes. De esta forma se empaquetan puntos de variación y variantes y se hace el modelo más manejable. Por ejemplo, en los STO se pueden definir paquetes como (a) *LimpiezaBásicaCompleta*, que contendría las variantes *Limpieza, Limpieza Completa* y *Teleoperado* y (b) *LimpiezaBásicaMancha*, que contendría las variantes *Limpieza, Mancha* y *Teleoperado*. La notación para definir estos puntos de variación abstractos y sus alternativas es la misma que la mostrada en la Figura 20.

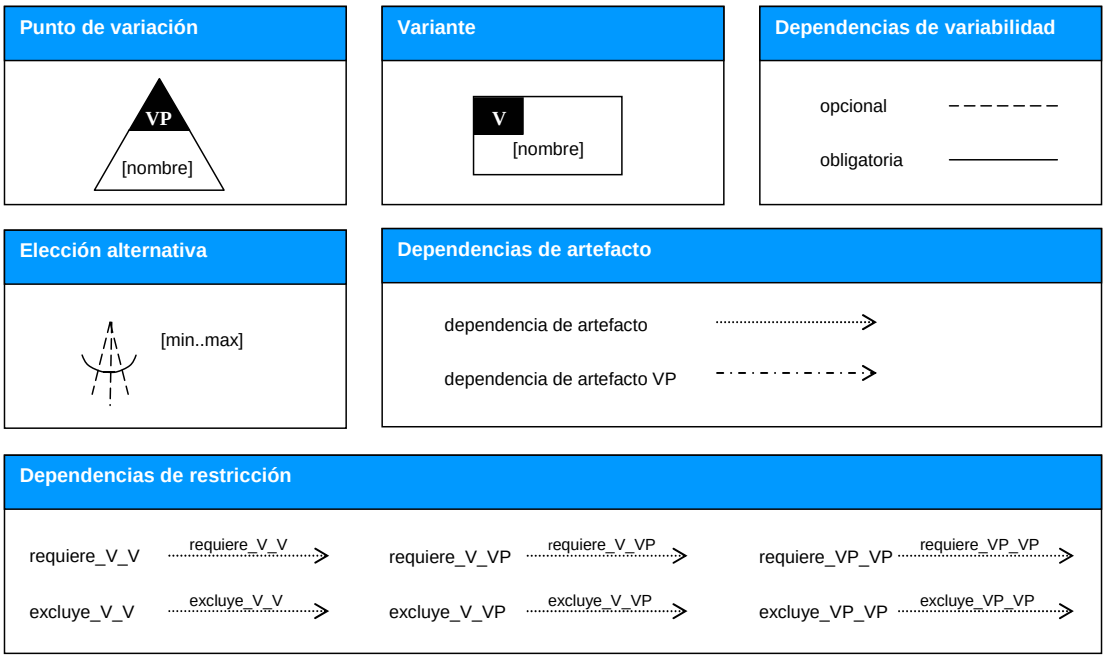

Figura 20 Representación gráfica de la variabilidad (adoptado de Pohl *et al.* [249])

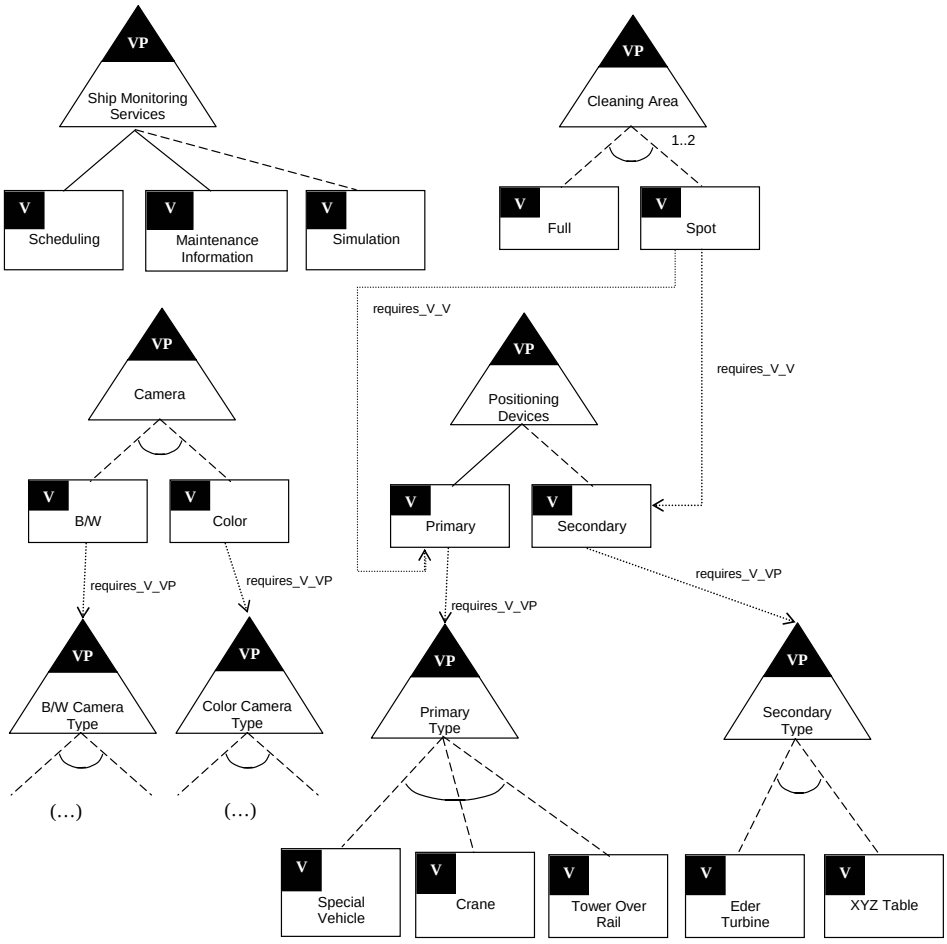

Figura 21 Diagrama de variabilidad ortogonal de los STO (extracto)

3.3.2.5 Modelo de variabilidad ortogonal y variabilidad en los requisitos

El modelo de variabilidad ortogonal permite especificar la variabilidad de manera uniforme en los diferentes modelos del desarrollo sin tener que modificar las notaciones originales. La idea esencial es relacionar una variante en el modelo de variabilidad con el subconjunto de cada uno de los modelos del desarrollo que está relacionado con la especificación de dicha variante (a través de las correspondientes dependencias de artefacto).

En esta sección se sigue a Pohl *et al.* [249] cuando muestra cómo el modelo de variabilidad ortogonal permite documentar la variabilidad en los requisitos textuales, modelos de características y casos de uso. Por cuestiones de espacio la explicación es breve y no se incluyen otros modelos de requisitos que Pohl *et al.* [249] también tratan y que llaman *tradicionales* (diagramas de flujo de datos, diagramas de clase y diagramas de máquinas de estado).

Variabilidad en los requisitos textuales

Los requisitos textuales expresan la variabilidad por medio de ciertas frases recurrentes o palabras clave. Para que la variabilidad sea soportada de forma no ambigua en los requisitos textuales se requiere su modelado explícito. Por ejemplo, supongamos que en la especificación de los STO se tiene el siguiente requisito: "El sistema teleoperado deberá disponer de dispositivos primarios y secundarios que permitan posicionar la herramienta con la suficiente precisión sobre la superficie del casco (entre 5 y 15 cms.)". La oración subordinada ("que permiten…") introduce en este caso una pequeña ambigüedad (¿se refiere sólo a "secundarios" o a "primarios y secundarios"?). Esta ambigüedad quedaría eliminada al representar explícitamente la variabilidad en el requisito textual tal y como proponen Pohl *et al.* [249] –ver Tabla 33–. Esta información se puede representar mediante texto enriquecido con etiquetas XML.

Punto de variación	*El sistema teleoperado deberá disponer de dispositivos…*
Variante	*…primarios…*
Variante	*…y secundarios…*
	…que permitan posicionar la herramienta con la suficiente precisión sobre la superficie del casco (entre 5 y 15 cms.)

Tabla 33 Ejemplo de representación de la variabilidad en requisitos textuales

Variabilidad en el modelo de características

Como hemos visto en la Sección 3.3.2.1, el modelo de características modela la variabilidad del sistema haciendo uso de una notación parecida a la del modelo de variabilidad ortogonal. Pohl *et al.* [249], además de otros autores, argumentan que los árboles de características como los que proponen FODA y FORM tienen varias limitaciones a la hora de representar la variabilidad:

- Con un árbol de características no se distingue entre características alternativas que son comunes a todas las aplicaciones (y por tanto son comunes a la línea de productos) y características alternativas que han de ser seleccionadas para una determinada aplicación dentro de la línea. Por ejemplo, el sencillo caso de la

Figura 22 tiene tres posibles interpretaciones: (a) Cada STO debe disponer de exactamente uno de los dos tipos de herramienta; (b) Cada STO debe disponer de los dos tipos de herramienta, y es el operador el que selecciona uno u otro en el momento adecuado; (c) Cada STO debe disponer de uno de los tipos de herramienta, o bien de los dos y permitir al operador que seleccione uno u otro. Las opciones (a) y (c) indican que el árbol de características prevé variabilidad en el tipo de herramienta, pero tienen interpretaciones diferentes que llevarían al desarrollo de dos sistemas distintos. La interpretación (b), en cambio, indica que el tipo de herramienta es un elemento invariante, común a todos los STO.

- El árbol de características no tiene un mecanismo de agrupación que permita que un conjunto de características arbitrario sea asignado a una variante. Por ejemplo, los paquetes de opciones que se mencionan en el último párrafo de la Sección 3.3.2.4 no son de aplicación a los árboles de características, pues cuando involucraran a características de ramas distintas requerirían la reestructuración del árbol, de manera que la descomposición original del modelo se perdería.

Pohl *et al.* [249] prefieren combinar el uso del modelo de variabilidad ortogonal con el modelo de características, seleccionando partes del árbol de características al elegir una variante en la resolución de un punto de variación en el modelo de variabilidad. Una misma característica puede ser trazada desde distintas variantes del modelo de variabilidad.

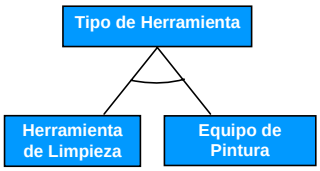

Figura 22 Características alternativas con la notación original de FODA/FORM

Bühne *et al.* [48] profundizan en la descripción de los problemas de los árboles de características en el modelado de líneas de productos, a partir de su experiencia en el dominio de la automoción, y muestran cómo el modelo de variabilidad ortogonal soluciona tales problemas.

Variabilidad en el modelo de casos de uso

Como se ha comentado con anterioridad, la variabilidad en los casos de uso puede aparecer en los diagramas de casos de uso, en los escenarios –descritos por ejemplo con un diagrama de secuencia UML–, o en la descripción textual del caso de uso a través de una plantilla. Pohl *et al.* [249] describen cómo el modelo de variabilidad ortogonal puede servir para asegurar la consistencia en la variabilidad que existe en los distintos tipos de documentación del caso de uso (Figura 23).

3.3.3 Un modelo de procesos general para la ingeniería de requisitos del dominio

Greenfield y Short [125] ejemplifican el proceso de desarrollo de una factoría de software (ver Figura 24). En esta sección se resume la visión que su propuesta proporciona sobre la IR del dominio y de la aplicación. Aunque los procesos se

describen secuencialmente, en realidad se realizan de forma iterativa e incremental. El desarrollo de una línea de productos comienza con un proceso de análisis, que se refina en los siguientes subprocesos: (1) Definición de la línea de productos; (2) Alcance del dominio del problema; y (3) Alcance del dominio de la solución.

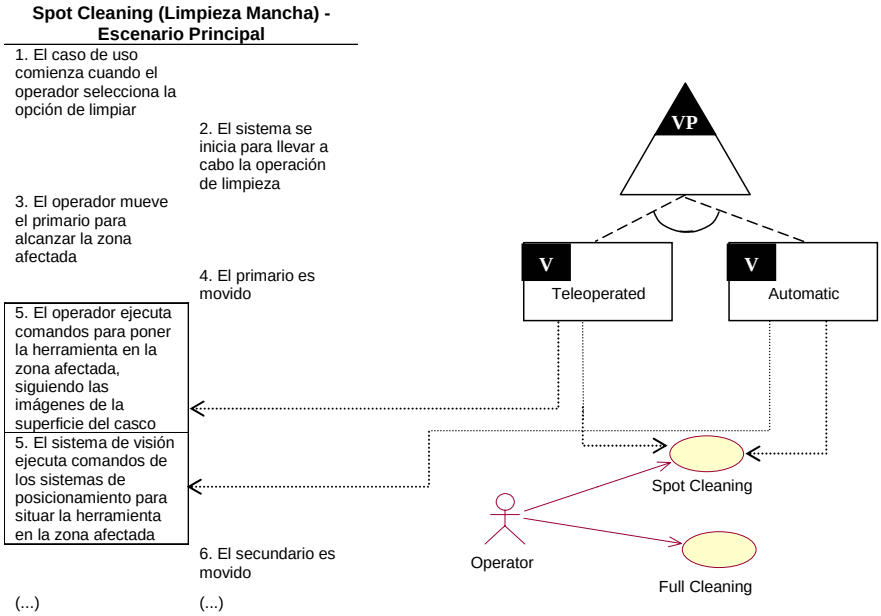

Figura 23 Ejemplos de dependencias de artefacto con el modelo de casos de uso

La definición de la línea de productos da lugar a la *descripción del dominio del problema*, como una lista de descripciones textuales de cuestiones de alto nivel que han de ser resueltas con los productos de la línea de productos. También se identifican los procesos del negocio implicados, que se describen textualmente en unas pocas líneas.

Después se determina el alcance del dominio del problema, que da lugar a un *modelo de características del problema*, que describe las características comunes y variables en el dominio del problema. Se puede realizar también un modelo de procesos del negocio, en el cual no es preciso entrar en muchos detalles: los objetivos son comprobar que el modelo de características describe correctamente el dominio del problema y proporcionar un contexto.

Posteriormente, el alcance del dominio de la solución da lugar a un *modelo de características de la solución*, que describe las características comunes y variables de la línea de productos, no ya del dominio del problema (la Figura 17 es un ejemplo de un modelo de este tipo). El modelo de características de la solución ha de incluir características requeridas por la implementación que no aparecieron en el alcance del dominio del problema, como por ejemplo características para la administración de almacenes de datos. Se diferencia del modelo de características del problema básicamente en los siguientes aspectos:

- El modelo de características del problema describe problemas que pueden ser resueltos por varios sistemas, mientras que el modelo de características de la solución describe sólo la parte de la solución proporcionada por esta línea de productos.

- El modelo de características del problema puede describir entidades físicas (como personas) y procesos ejecutados manualmente, mientras que el modelo de características de la solución describe sólo entidades lógicas y procesos automatizados.
- Las entidades o procesos descritos por el modelo de características de la solución pueden contener propiedades o pasos distintos de los descritos por el modelo de características del problema, pues el modelo de características de la solución describe posibles implantaciones de las entidades y procesos del problema.

En el diseño de la línea de productos, una actividad clave es la asignación de requisitos (ver Figura 24), cuyo objetivo es asignar las variabilidades en los requisitos de la línea a los mecanismos para gestionar la variabilidad en la plataforma, especialmente en la arquitectura y en los componentes de implementación. Para ello se define una matriz en la que se relacionan las características y los assets de la plataforma de la línea de productos. Cada celda (i,j) indica cómo implementar la característica F_i con el asset A_j: herramientas que se deben usar, pasos de desarrollo a seguir, etc.

A partir del modelo de características de la solución se derivan los requisitos de la línea de productos, que constituyen una representación utilizada durante el desarrollo del producto para indicar qué características opcionales deben ser incluidas en un producto de la línea. Durante la especificación del producto se debe ayudar a los clientes a expresar sus requisitos en términos de los requisitos de la línea de productos. Como resultado de la especificación del producto, los assets de la plataforma que soportan las características comunes y variables seleccionadas se pueden cargar en el espacio de trabajo compartido por los desarrolladores del producto en la factoría de software.

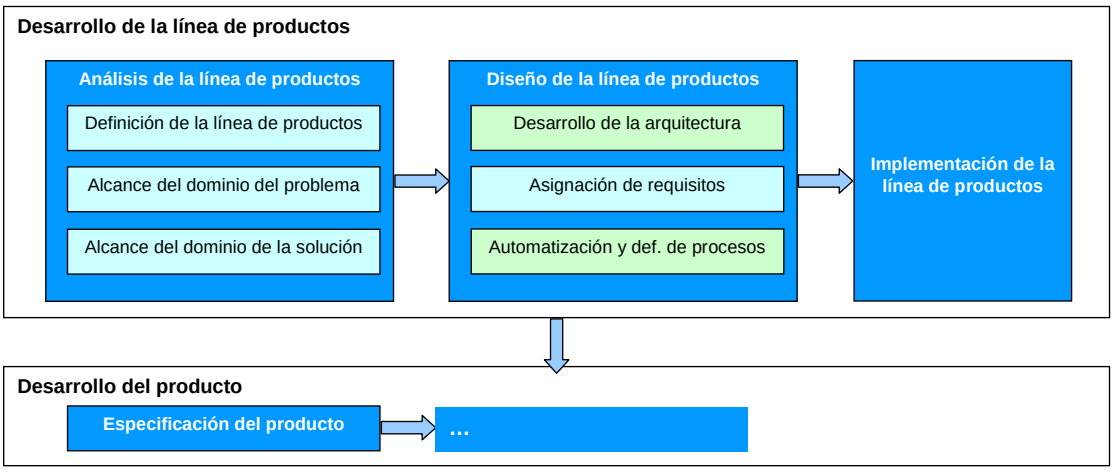

Figura 24 Esquema de una factoría software (adaptado de Greenfield y Short [125])

3.4 Propuesta integrada para el análisis de los sistemas teleoperados

En esta sección seguimos en buena medida el artículo de Nicolás *et al.* [225], que culmina una serie de resultados parciales anteriores [176, 222-224], para dar cuenta de la propuesta diseñada para realizar el análisis del dominio de los STO y las lecciones aprendidas en dicho caso de estudio. El método de investigación escogido, I-A (Sección 3.4.1) nos ha llevado a buscar soluciones prácticas –la propuesta descrita a continuación, que denominamos SIRENspl– a problemas concretos –el análisis del dominio de los STO–. Es por ello que propuesta y caso de estudio están íntimamente ligados y en esta Sección 3.4 se presentan conjuntamente, con la intención de trasmitir adecuadamente el conjunto de decisiones que se han adoptado frente a los problemas encontrados en el dominio. Sin embargo, como se manifiesta en las conclusiones de este capítulo (Sección 3.6), creemos que la propuesta de SIRENspl puede ser extrapolada a otros sistemas teleoperados.

SIRENspl se construye sobre la base de la revisión general del estado del arte presentada en la Sección 3.3. Como veremos en las secciones posteriores, la decisión más importante ha sido basar el modelo de variabilidad en un modelo de características, con una serie de extensiones fruto de un estudio más profundo del estado del arte, mostrado en la Sección 3.4.3. El modelo de características diseñado, si bien no tiene la potencia expresiva completa del modelo de variabilidad ortogonal presentado en la Sección 3.3.2.4, ha sido suficiente para especificar de forma sencilla la variabilidad en este caso de estudio. El modelo de casos de uso diseñado para esta experiencia también se deriva de un estudio más profundo del estado del arte, mostrado en la Sección 3.4.4.

3.4.1 Programación de la investigación

Investigación en acción (I-A, ver Sección 1.3.2) es un método de investigación cualitativa que puede ser utilizado para estudiar los efectos de cambios en los métodos de desarrollo de software. Los procesos sociales complejos, como el uso de tecnologías de la información, se pueden estudiar de forma adecuada mediante I-A, introduciendo cambios en tales procesos y observando los efectos de tales cambios. I-A promueve un proceso de aprendizaje reflexivo y una búsqueda de soluciones prácticas que involucra tanto a investigadores como a desarrolladores.

En línea con la terminología de I-A, en el caso de estudio de los STO se han utilizado los siguientes roles:

- El investigador es el Grupo de Investigación en Ingeniería del Software de la Universidad de Murcia (GIS).
- Lo investigado, es decir, el objeto bajo estudio, es la línea de productos de los STO.
- El Grupo Crítico de Referencia (GCR), es decir aquel que tiene el problema que se intentan resolver, es la División de Sistemas e Ingeniería Electrónica (DSIE)

de la Universidad Politécnica de Cartagena (UPCT). La DSIE realiza transferencia de tecnología a compañías navales como Navantia (anteriormente Izar), interesadas en la innovación que los sistemas teleoperados proporcionan al mantenimiento de cascos de buques. Es de destacar que en dicho proceso la DSIE está directamente encargada del desarrollo de tales robots, en su totalidad o en parte.

- Los interesados son todas aquellas organizaciones que se pueden beneficiar de los resultados de la investigación: en este caso, el GCR y, en general, empresas que realizan tareas de mantenimiento de cascos de buques o manejan sistemas teleoperados similares.

Este caso de estudio ha sido desarrollado en el contexto de DYNAMICA, un proyecto de investigación de tres años descrito en la Sección 1.4.2.2. En el ámbito del proyecto se presentaron y discutieron informes del trabajo en curso en cinco talleres generales, de dos días de duración. Se produjeron además cuatro encuentros específicos, de una jornada, entre el investigador y el GCR. El grueso de la comunicación entre el investigador y el GCR fueron comunicaciones personales, normalmente por email, que no fueron registradas formalmente.

En este caso de estudio ha tenido lugar una aplicación *participativa* de I-A, en la cual el GCR pone en práctica las recomendaciones propuestas por el investigador, con el cual comparte los efectos y los resultados. Para cuantificar el valor que la propuesta presentada en este capítulo ha tenido para el GCR se han usado cuestionarios. Una variante *empírica* de I-A hubiera sido difícil de aplicar en este caso de estudio porque habría requerido que el GCR hubiera realizado un registro sistemático de acciones y efectos, pero como se explica en la Sección 3.4.7, no existían registros del trabajo previo del GCR.

En la Figura 1 de la página 38 de esta memoria de tesis doctoral se describen las actividades realizadas en los ciclos iterativos de I-A. El ciclo comienza con una *planificación*, en la cual se identifican las cuestiones que guían la investigación y se especifican las acciones para resolver estas cuestiones. En este caso de estudio, en primer lugar, surgió el interés de realizar un análisis del dominio de los STO por las razones expuestas en la Sección 2.1. El estado del arte en el campo del análisis del dominio fue estudiado después con el fin de adoptar o adaptar los modelos de análisis necesarios para afrontar el problema. El estudio del dominio fue facilitado por el hecho de que el GCR tenía experiencia en el desarrollo de STO y existía un amplio conjunto de documentos disponibles que describían los objetivos principales y las necesidades de estos sistemas, su arquitectura de referencia, la especificación de los comandos utilizados en su desarrollo, algunas cuestiones informales sobre seguridad a terceros (*safety*) y seguridad física (*security*), y una documentación inicial de requisitos, incluyendo un modelo conceptual inicial, un modelo FODA [160] inicial de características para una parte de la línea de productos y un intento inicial de identificación de casos de uso. Toda esta documentación ayudó al investigador a ganar conocimiento del dominio con rapidez.

Después de la planificación, se prosigue con una actividad de *acción* en la cual el investigador induce una variación de la práctica cuidadosa, deliberada y controlada. En este caso de estudio el investigador y el GCR trabajaron juntos iterativamente de forma

continuada para establecer un modelo del dominio de los STO utilizando las técnicas seleccionadas y/o adaptadas en la actividad anterior.

A continuación se realiza una *observación* o *evaluación*, en la cual se recoge información sobre los efectos de la acción realizada, y algunas lecciones aprendidas comienzan a tomar forma. Esta observación fue realizada de forma completamente cualitativa en las primeras iteraciones, y finalmente fue soportada por el análisis de un cuestionario de 20 puntos que se dirigía a evaluar las cuestiones que han sido consideradas de interés en la primera valoración cualitativa.

Finalmente, el ciclo finaliza con una *reflexión*, en la cual los resultados son compartidos y analizados por todos los interesados, y pueden surgir nuevas cuestiones para ser tratadas en un segundo ciclo, y así sucesivamente. En general esta reflexión ha sido realizada independientemente por el investigador y el GCR y finalmente se alcanzó un consenso.

3.4.2 Un marco para el análisis del dominio de los STO

En la etapa de planificación del caso de estudio mediante I-A ha sido necesario seleccionar y caracterizar los modelos a utilizar para realizar el análisis del dominio de los STO. El trabajo de Olivé [232], que describe la naturaleza de los esquemas conceptuales y el rol que juegan en el desarrollo de sistemas de información, ha sido utilizado para unificar la terminología utilizada y proporcionar un marco sencillo para ubicar las técnicas utilizadas en este caso de estudio. Olivé se centra en sistemas de información, pero creemos que sus conclusiones se pueden aplicar en el campo general del desarrollo de software. Siguiendo a Olivé se puede definir *esquema conceptual* como el modelo de conocimiento que un sistema necesita para realizar sus funciones. Con la presente experiencia se trata de construir, por tanto, un esquema conceptual de los STO.

Existe cierta confusión en la literatura en relación con las similitudes y diferencias entre la noción de esquema conceptual y conceptos similares, como modelo del dominio (*domain model*), conocimiento del dominio (*domain knowledge*), especificación funcional (*functional specification*) y ontología (*ontology*). El trabajo de Olivé liga distintas ideas ya existentes con el objetivo de (1) identificar la naturaleza del esquema conceptual y el rol que juega en el desarrollo de sistemas de información; y (2) mostrar sus correspondencias con términos relacionados. Olivé considera que el esquema conceptual de un sistema consiste de dos tipos de conocimiento: (1) conocimiento sobre el dominio; y (2) conocimiento sobre las funciones que el sistema realiza. El primero se denomina *Esquema Conceptual del Dominio* (ECD, *Domain Conceptual Schema*) y el segundo *Especificación Funcional* (EF, *Functionality Specification*). La Figura 25 muestra cómo se han establecido estos modelos en este caso de estudio, donde las líneas punteadas representan relaciones de traza.

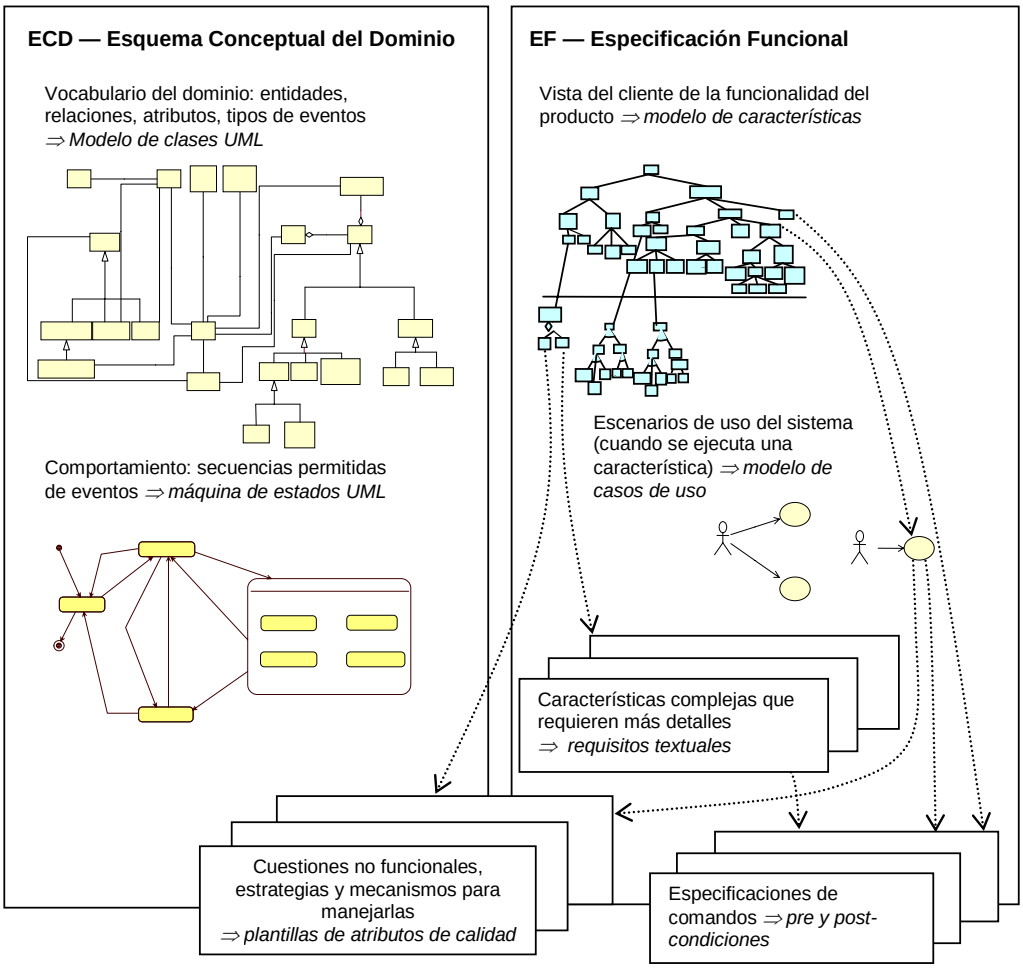

Figura 25 Modelos utilizados en el análisis del dominio de los STO

El ECD, a veces denominado *conocimiento del dominio* (*domain knowledge*) o *modelo del dominio* (*domain model*), ha sido implementado en este caso de estudio por medio de las siguientes técnicas:

- Un modelo de clases UML, que muestra (1) una taxonomía de *tipos de entidades del dominio* (con sus atributos, relaciones, y restricciones de integridad); y (2) un conjunto de *tipos de eventos del dominio*. Por ejemplo, este modelo incluye entidades como *Tool*, *Blasting*, *Primary* y *Secondary*, y tipos de eventos como aquellos que provocan que el sistema cambie de un estado de mantenimiento a otro (*Apprenticeship*, *Calibration*, *Diagnosis* y *Configuration*). El modelo conceptual consiste de alrededor de 80 clases conceptuales.
- Una máquina de estados UML, que describe la dinámica de los tipos de eventos del dominio. En este caso de estudio sólo ha sido necesario modelar una máquina de estados para describir las secuencias de eventos que cambian el estado del sistema (ocho estados).

La EF, por otro lado, ha sido implementada en este caso de estudio por medio de las siguientes técnicas:

- Un modelo de características (*feature model*), que creemos que especifica de forma muy intuitiva la visión que tienen los interesados de la línea de productos. El modelo completo recoge unas150 características.
- Un modelo de casos de uso genéricos (*generic use case model*), que describe los escenarios de interacción entre los actores y el sistema que se producen durante la ejecución de lo que denominamos *características funcionales* (es decir, de las características que capturan una función ejecutable de la línea de productos). Este modelo consiste de unos 20 casos de uso.
- Una colección de requisitos textuales, que detallan cuestiones más detalladas de la línea de productos que no pueden ser incluidas adecuadamente en las plantillas de las características o en los casos de uso. Se han especificado alrededor de 550 requisitos textuales.
- Una colección de especificaciones de comandos, que son básicamente contratos con pre y postcondiciones que describen los comandos utilizados en la implementación de los STO. Existen alrededor de 25 comandos (principalmente comandos de calibración y de movimiento).

Las cuestiones no funcionales han sido modeladas a través de plantillas de atributos de calidad (*quality attribute templates*) [36]. Estas plantillas no sólo abarcan requisitos, sino que también ayudan a ligar los requisitos con la arquitectura de diseño. Una parte de estas plantillas se corresponde con el ECD, cuando se describen estímulos del sistema que están ligados con cuestiones no funcionales (estímulos que se denominan *escenarios abstractos* en este contexto). La otra parte de estas plantillas se corresponde con la EF, cuando se describen las estrategias y mecanismos necesarios para resolver tales escenarios. Es debido a esta dualidad por lo que las plantillas de atributos de calidad se muestran en la Figura 25 a caballo entre el ECD y la EF. Por ejemplo, los escenarios abstractos están ligados a cuestiones como (1) adaptabilidad (describiendo cambios en los elementos del sistema, como la clase de herramienta utilizada para hacer la limpieza o sobre el uso de un nuevo sistema operativo); (2) rendimiento (describiendo los cambios en el tiempo de proceso de las tareas); o (3) disponibilidad (tratando con las condiciones de una parada segura o con cuestiones sobre la disponibilidad de las comunicaciones). Cuando estos escenarios abstractos se relacionan con la seguridad o la seguridad a terceros realmente se pueden considerar *amenazas*, y la línea de productos debe minimizar los riesgos asociados con estos escenarios abstractos: por ejemplo, cuando el sistema entra en un estado en el que no es seguro ejecutar ciertos comandos. En la especificación de los STO se han incorporado alrededor de 20 plantillas de atributos de calidad, que son de grano grueso.

En la línea de productos de los STO, como se muestra en las siguientes secciones, la variabilidad ha sido capturada (1) en el modelo de características; (2) en los casos de uso; y (3) en las plantillas de atributos de calidad. Con esta propuesta, el punto central donde se captura la variabilidad de la línea de productos es el modelo de características. En un extenso trabajo sobre formas de realizar la variabilidad, Svahnberg *et al.* [278] afirman que existe un cierto consenso en que la variabilidad del dominio puede ser identificada más fácilmente si el modelado del dominio se basa en el uso de características. Sin embargo, aunque el modelo de características proporciona potentes mecanismos para plasmar la variabilidad de la línea de productos, la variabilidad no se circunscribe únicamente al modelo de características. Puede haber también variabilidad en la forma de solventar una amenaza sobre el sistema (dentro de una plantilla de

atributos) y en los escenarios típicos de uso de distintos productos dentro de la misma línea de productos. En este sentido, existe un consenso (ver por ejemplo [122]) en cuestionar la idoneidad de los casos de uso convencionales para el modelado de líneas de productos, introduciendo de una u otra manera el concepto de variabilidad en los casos de uso, que a veces son referidos como casos de uso *genéricos* [155]. En un caso de uso genérico, algunos pasos pueden variar dependiendo del producto concreto de la familia de productos o de ciertos parámetros. De este modo, el modelo de casos de uso genéricos es el que recoge aquella variabilidad exclusiva a los escenarios de uso del sistema. Al repartir la variabilidad de esta manera entre los modelos de características y de casos de uso se pretende obtener modelos más simples y evitar duplicidad de información en los modelos de características y de casos de uso, que son complementarios. En la Figura 26 se presenta un metamodelo inicial de los requisitos diseñado para la presente experiencia, cuyos detalles se explican en las Secciones 3.4.3, 3.4.4, 3.4.5 y 3.4.6.

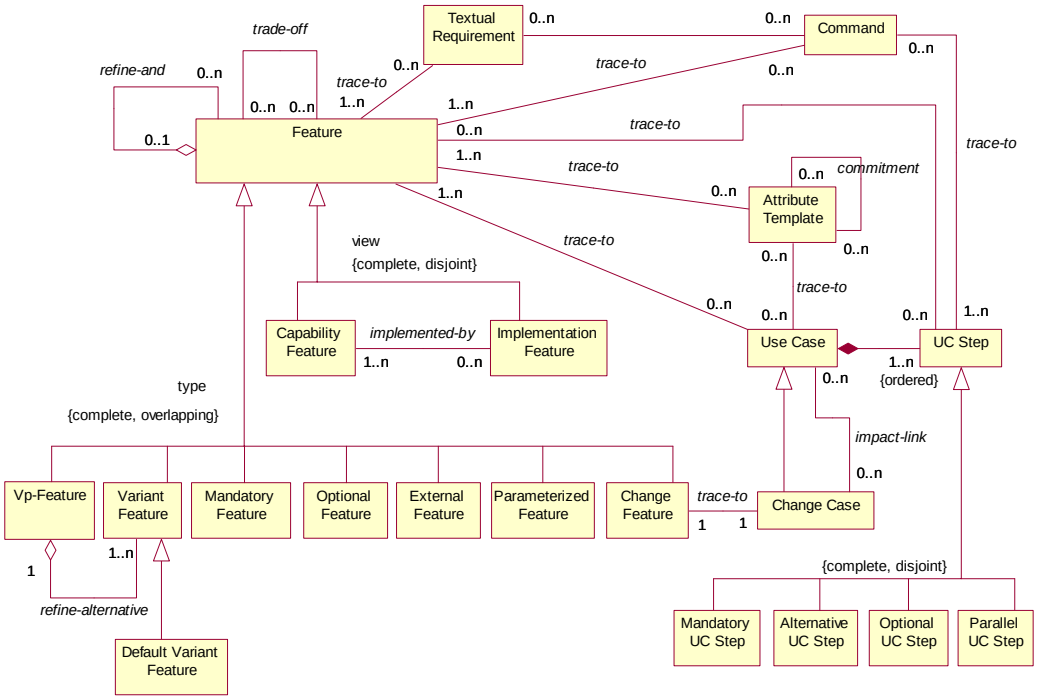

Figura 26 Metamodelo de requisitos inicial diseñado para el análisis del dominio de los STO

El proceso de desarrollo de los modelos del dominio de los STO ha sido realizado de la siguiente manera. En primer lugar se desarrolló el ECD para obtener un conocimiento del dominio compartido por parte del investigador y del GCR. Posteriormente, tras una revisión de la literatura se seleccionaron los modelos para describir la EF, que fueron construidos iterativamente. La decisión más importante de la planificación del caso de estudio fue centrar el modelado de la EF en un modelo de características. El modelo de características fue así seleccionado primero para expresar las funcionalidades comunes y variables de la línea de productos. Trabajando en el modelo de características de los STO se determinó que algunas características necesitaban ser extendidas por medio de requisitos textuales, que no se ajustan adecuadamente a las plantillas de características y

que proporcionaban el nivel de detalle adecuado. Trabajando también en el modelo de características se hizo evidente que la ejecución de ciertas funciones de la línea de productos conducía a la ejecución de escenarios de interacción que debían ser descritos de alguna manera: se consideró que un modelo de casos de uso, por tanto, podía ser útil. Mientras se comenzaba a construir el modelo de casos de uso, pronto se vio que los casos de uso convencionales no podían dar cuenta de toda la variabilidad de la línea de productos y que necesitaban ser extendidos. Se determinaron por tanto las extensiones de los casos de uso tras un nuevo estudio de la literatura, y los modelos de características y de casos de uso genéricos se desarrollaron concurrentemente. Finalmente, se consideró conveniente incluir las plantillas de atributos de calidad entre los modelos de análisis del dominio. Estas plantillas, que ya estaban disponibles como un producto de la arquitectura de referencia, fueron introducidas para tratar con requisitos no funcionales. El paso que daban hacia el diseño y la consiguiente ligadura entre el análisis del dominio y la arquitectura de referencia se consideró interesante.

3.4.3 Capacidades y restricciones del sistema mediante características

Como se comenta en la Sección 3.3.2.1, en el modelo de características se especifican las capacidades (funcionales y no funcionales) y las restricciones tecnológicas de los productos de la línea de productos. En este caso de estudio se ha adaptado la notación del modelo de características definido por Kang *et al.* en FORM [161]. Se ha elegido FORM como base de la propuesta ya que creemos que es un método de diseño de líneas de productos maduro y bien conocido, que se ha aplicado en varios casos de estudio publicados (por ejemplo [162, 181]). Además, FORM soporta específicamente el proceso de IR, en contraste con otros métodos de diseño de la arquitectura de líneas de productos [210]. El modelo de características original de FORM ha sido adoptado y extendido para este caso de estudio como se indica en los párrafos siguientes.

FORM organiza las características en cuatro niveles [161, 182]: *capacidad*, *entorno operativo*, *tecnología del dominio* y *técnica de implementación*. Con el objetivo de obtener un modelo de características más sencillo, hemos reducido estos cuatro niveles a solamente dos: *capacidad* e *implementación*. Este último, por tanto, reúne los niveles originales de entorno operativo, tecnología del dominio y técnica de implementación, que en esencia están muy relacionados y que, en nuestra opinión, en la práctica a veces parecen solaparse dando lugar a confusión. En esta línea, Trigaux y Heymans [286], por ejemplo, también critican la complejidad de estos cuatro niveles. En la Figura 27 se puede observar un extracto del diagrama de características, donde se refleja parte de las características asociadas a los servicios ofrecidos por la línea de productos y a ciertas cuestiones no funcionales (el extracto es el mismo que el de la Figura 17, si bien la notación de la Figura 27 incluye las extensiones que se comentan en esta sección). Las relaciones entre los niveles de capacidad y de implementación se especifican a través de trazas *implemented-by*, como por ejemplo la que muestra los tipos de tecnología utilizada para realizar la limpieza, *Grit-Blasting* (*Limpieza basada en granalla*) e *Hydro-Blasting* (*Limpieza basada en agua a presión*).

von der Maβen y Lichter [289] establecen los requisitos que debe satisfacer cualquier notación para modelar la variabilidad en una línea de productos. Posteriormente, Trigaux y Heymans [286] extienden dichos requisitos con uno más: la representación gráfica de la variabilidad, y en concreto, de los puntos de variación, de las variantes y de

las cardinalidades de los puntos de variación. Con el objetivo de satisfacer estos requisitos para obtener una notación más expresiva, hemos extendido la notación original de FORM con la representación gráfica de los puntos de variación y de las cardinalidades:

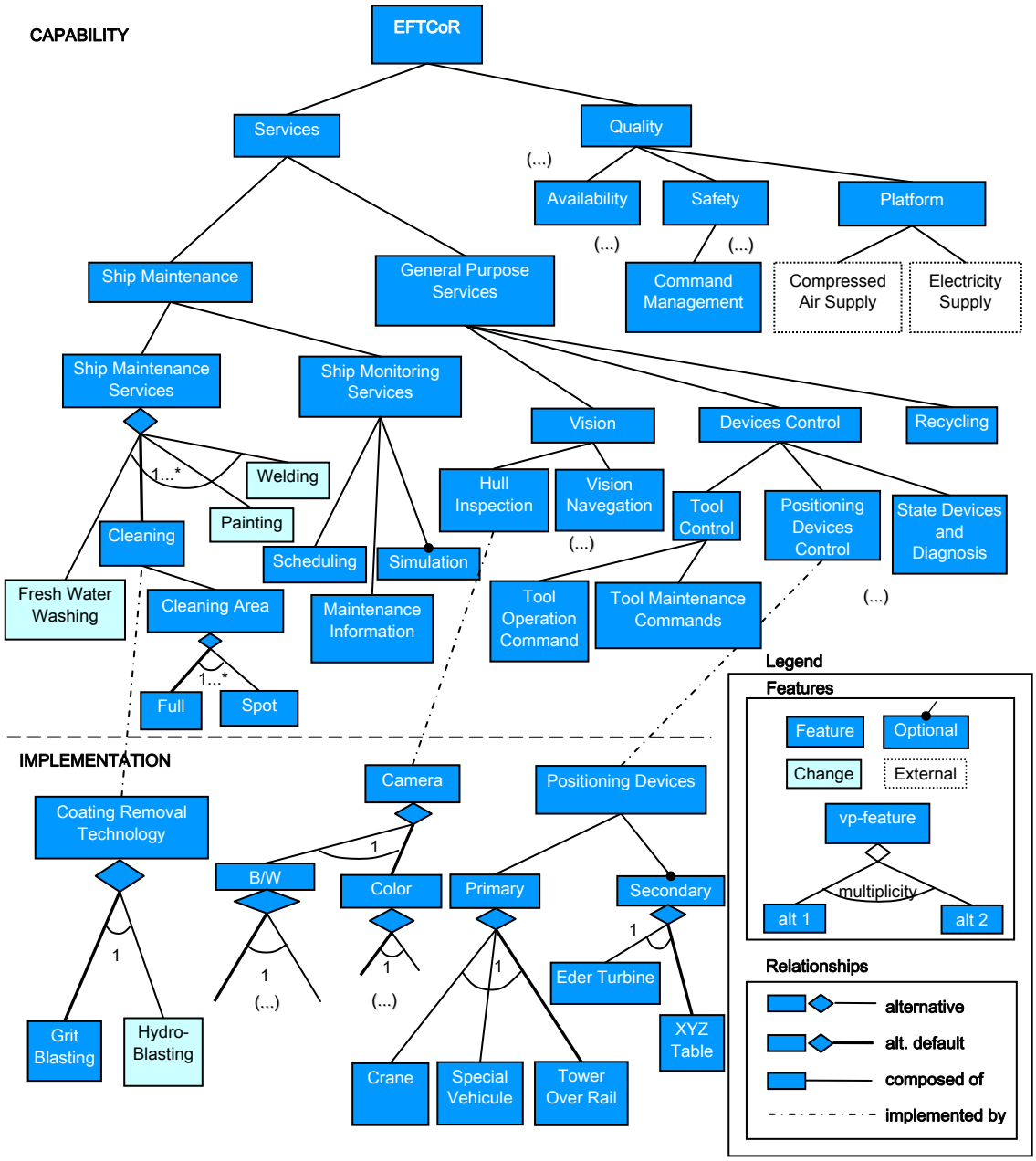

Figura 27 Un extracto del modelo de características

- En la representación gráfica de los puntos de variación, en línea con Griss *et al.* [126], consideramos que una característica puede representar un punto de variación (denominándose *vp-feature*) mientras que otras características juegan el papel de sus variantes (*variant features*). Por ejemplo, en la Figura 27 se observa como una relación *implemented-by* enlaza la característica *Hull Inspection (Inspección del casco)* con un punto de variación (vp-feature) dentro de la línea de producto, identificado como *Camera*, que puede ser *B/W (Blanco y negro)* o *Color*. Una característica puede además estar ligada a más de un punto

de variación: por ejemplo, en la Figura 27 se observa como la limpieza del casco (*Cleaning*) está ligada a dos puntos de variación: la superficie implicada (*Cleaning Area*) y la tecnología implicada (*Coating Removal Technology*).

- En cuanto a la representación gráfica de las cardinalidades, hemos adoptado la aproximación de Riebisch *et al.* [261], basada en la notación de UML, dado que nos ha parecido intuitiva, sencilla y completa (ver Figura 27). Es completa pues cubre todas las combinaciones posibles de cardinalidades. Sin embargo, para describir este dominio ha sido suficiente con distinguir nodos XOR (cardinalidad 1, se escoge una sola de las variantes) y nodos OR (cardinalidad 1..*, se escoge al menos una de las variantes).

Adoptamos el concepto de *característica externa* (*external feature*) que proponen Svahnberg *et al.* [278]. Estas son características proporcionadas por la plataforma en la cual se va a desplegar el sistema. No forman parte del sistema, pero son importantes porque el sistema las usa y depende de ellas. En los STO, por ejemplo, el astillero debe proporcionar unos valores determinados de alimentación neumática (*Compressed Air Supply*) y de alimentación eléctrica (*Electricity Supply*), lo cual se ha modelado como sendas características externas (Figura 27). A través de las características externas se clarifican las dependencias del sistema con otros sistemas con los que debe interactuar, interacciones que suponen una fuente importante de problemas en los desarrollos software cuando no son bien comprendidas. La variabilidad en características externas puede motivar la inclusión de elementos en el software para gestionar dicha variabilidad.

A partir de las sesiones con los desarrolladores de la línea de productos, se ha detectado la necesidad de incluir un nuevo tipo de características (que denominamos *características de cambio*, *change features*) cuya inclusión en la línea de productos se prevé en un futuro, pero que todavía no están disponibles. Por ejemplo, en la línea de productos de los STO, en la Figura 27 se muestra el pintado del casco del buque (*Painting*) o la soldadura del mismo (*Welding*).

En este caso de estudio utilizamos parámetros globales y locales en la plantilla de las características parametrizadas (*parameterized features*) de FORM. Los caracteres $ y @ se utilizan como prefijo de las variables globales y locales, respectivamente. Como ejemplo de parámetros globales tenemos la resolución mínima de una cámara en el sistema y el tiempo de respuesta para un movimiento en el secundario. Una vez se asigna un valor al parámetro global, se instancia en toda la especificación. Como ejemplos de parámetros locales tenemos la resolución específica de una cámara (por ejemplo, la resolución de una cámara digital de color) y el tiempo de respuesta para un movimiento de una *Mesa XYZ* (*XYZ Table*).

La variabilidad de la línea de productos queda plasmada en el modelo de características a través de (1) la representación gráfica de los puntos de variación y de sus variantes, y de las características (2) opcionales; (3) externas; (4) parametrizadas; y (5) de cambio. De forma complementaria, cada característica se describe mediante la plantilla de la Tabla 34, que aporta más información que la representación gráfica y extiende aproximaciones anteriores [160, 161]. En la cabecera de la plantilla se puede indicar primero si se trata o no de una vp-feature; *view* sirve para diferenciar entre características de capacidad (*capability*) y de implementación (*implementation*); *commonality* diferencia si la característica es obligatoria (*mandatory*), alternativa

(*alternative*) u opcional (*optional*); finalmente se puede especificar si la característica es *external* o *change feature*. Todos los campos de la plantilla salvo el primero se cumplimentan opcionalmente. En la Tabla 35 se muestra a modo de ejemplo la descripción textual para la característica *Spot* de la Figura 27, que hace referencia a la capacidad para limpiar una zona concreta (una mancha) en el casco del barco.

[**FEAT**|**VP-FEAT**] *nombreCaracterística* (view, commonality [“,” **external**][“,” **change**])
Descripción: descripción textual de la característica, que incluye posibles parámetros locales y globales.
Motivación: especifica la motivación que ha llevado a incluir esa característica en la línea de productos.
Reglas de composición: relaciones jerárquicas de dependencia mutua, ya sean alternativas *OR* o de composición *AND*. La multiplicidad se expresa con la notación usual de UML: 0..*, 1..*, 1, etc. Cuando en una composición AND una de las características sea opcional se denotará por *opt (featureK)*. En una composición OR, una característica se puede marcar como la opción por defecto de la siguiente manera: *(multiplicityK) featureK (default).*
Compromisos: las relaciones de composición AND/OR no son suficientes para expresar todas las posibles dependencias en el modelo. Este campo sirve para especificar los compromisos (*trade-offs*) entre características: *requires(featureList) and/or excludes (featureList)*. Se puede añadir un comentario en lenguaje natural sobre la naturaleza del compromiso, que en ocasiones es más compleja que la simple inclusión o exclusión: a veces una característica favorece o condiciona, pero no incluye o excluye.
***Traza hacia**:* Requisitos y/o comandos que están ligados con esta característica, listados por sus identificadores únicos. Se puede ligar la característica con los casos de uso relacionados UC, con un hipervínculo que apunte a un diagrama de casos de uso que incluya los casos de uso anteriores, con pasos concretos dentro de casos de uso relacionados UCSTEP, con requisitos textuales REQ, con las plantillas de atributos de calidad relacionadas AT y con los comandos COMM que operacionalizan la característica.
***Implementa**:* Característica del nivel de capacidad a la que esta característica implementa.
***Implementada por**:* Característica/s del nivel de implementación que pueden implementar esta característica.
END-FEAT

Tabla 34 Plantilla textual para las características

Las relaciones de composición AND/OR no son suficientes para expresar todas las dependencias posibles entre las características, de manera que se requiere un campo *Compromisos* (*Trade-offs*, ver Tabla 34) para especificar las relaciones *requiere* (*requires*) y *excluye* (*excludes*) entre características. En la Tabla 35 se muestra un ejemplo de *requires*. Por otro lado, por ejemplo, se tiene que *Tower Over Rail excludes Eder Turbine*. Pero la naturaleza del compromiso puede ser más compleja que una simple relación de inclusión o exclusión: una característica puede favorecer (*favours*) o condicionar (*conditions*) otra. Por ejemplo, *Crane favours XYZ Table* y *Special Vehicle conditions XYZ Table* (dado que el primero se ha de adaptar específicamente si se utiliza el segundo en un robot concreto).

Una cuestión que se ha planteado en el diseño de la plantilla anterior es la posible inclusión del *tiempo de ligadura de las características* (*feature binding time*) en las vp-features. Esta cuestión nos lleva a otra mayor, el análisis de la ligadura de las características (*feature binding analysis*), que Lee y Kang [180] dividen en tres cuestiones: (1) qué características se ligan (*feature binding units*); (2) cuándo se ligan las características (*feature binding time*); y (3) cómo se ligan las características (*feature binding techniques*). El ámbito de este trabajo se ciñe a las dos primeras cuestiones:

- En cuanto a las *unidades de ligadura de características*, según Lee y Kang [180] las características no se instancian aisladamente, sino que en su instanciación es preciso tener en cuenta las relaciones de composición y las relaciones *requires* y *excludes*. Las características que pertenecen a la misma unidad de ligadura están relacionadas con un mismo servicio, y por tanto deben existir juntas para la correcta operación del servicio.
- En cuanto al *tiempo de ligadura de las características*, Svahnberg *et al.* [278], con una visión de ciclo de vida, distinguen cuatro momentos en el ciclo de vida en los cuales se puede instanciar un punto de variación: (1) en tiempo de derivación de la arquitectura; (2) en tiempo de compilación; (3) en tiempo de enlace; y (4) en tiempo de ejecución. En el dominio de los STO, los puntos de variación y las características parametrizadas y opcionales se instancian en tiempo de derivación de la arquitectura, por lo cual se ha decidido no incluir el tiempo de ligadura en la descripción de las características.

FEAT *Spot - Limpieza Mancha* (*capability*, *alternative*)
Descripción: Limpieza que se realiza sólo en puntos aislados de la superficie del casco del barco.
Motivación: La limpieza completa de la superficie del casco (*full blasting*) sólo se realiza cuando ésta está dañada en más de un 75%, pues implica un incremento en los costes de material abrasivo, pintura, personal, etc. En la mayor parte de las ocasiones se realiza una limpieza parcial (sólo de las zonas deterioradas – manchas o *spots*) para minimizar los costes de operación. El 80% de los barcos que entrar en el astillero necesitan este tratamiento (Limpieza Mancha, *spot blasting*).
Reglas de composición: —
Compromisos*: requires* (*Primario*, *Secundario*). Es preciso tener sistemas de posicionamiento primario y secundario, pues el sistema de posicionamiento primario no es suficientemente preciso por sí mismo.
Traza hacia*: UC Limpieza Mancha - Spot Cleaning, REQ101, REQ102, (…)*
Implementa*: —*
Implementada por*: —*
END-FEAT
REQ101: El sistema deberá realizar limpieza mancha cuando menos del 75% del casco del buque esté dañado.
REQ102: Limpieza mancha implica el uso de sistemas de posicionamiento primario y secundario.
REQ103: El sistema deberá forzar que el movimiento del sistema de posicionamiento primario implique que el secundario esté parado.
REQ104: El sistema deberá forzar que el movimiento del sistema de posicionamiento secundario implique que el primario esté parado.

Tabla 35 Plantilla para la característica *Spot* y ejemplos de requisitos textuales relacionados

3.4.4 Escenarios de interacción mediante casos de uso genéricos

En este caso de estudio, la ejecución de la funcionalidad representada por ciertas características se puede especificar de forma natural como un caso de uso o una combinación de casos de uso: cuando un actor requiere la ejecución de la funcionalidad representada por una característica para conseguir un resultado observable de valor,

dando lugar a un conjunto de escenarios de interacción con un producto de la línea de productos (por ejemplo, obsérvese la característica *Cleaning* de la Figura 27). Como se comenta en la Sección 3.3.2.2, los casos de uso convencionales no son suficientes para soportar la variabilidad de una línea de productos, pues describen las acciones de un actor cuando se realiza una cierta tarea mientras se interacciona con un sistema determinado. Sin embargo, el modelado de una línea de productos requiere la descripción de tareas análogas para diferentes productos en la línea de productos, esto es, requiere de casos de uso *genéricos* o de *línea de productos*. La especificación de las partes comunes y variables de las interacciones con los productos de la línea de productos debe estar descrita en los casos de uso para que estos sean adecuados para el modelado de líneas de productos. No existe un único formalismo ampliamente aceptado para integrar el modelado de la variabilidad en los casos de uso de la línea de productos. En esta sección se describe el enfoque adoptado en este caso de estudio después de realizar una revisión del estado del arte.

Se utilizan diagramas de casos de uso para agrupar visualmente los casos de uso que están involucrados en la ejecución de la funcionalidad representada por una característica especialmente compleja o importante. Por ejemplo, la Figura 28 muestra los casos de uso relacionados con la limpieza del casco (característica *Cleaning*).

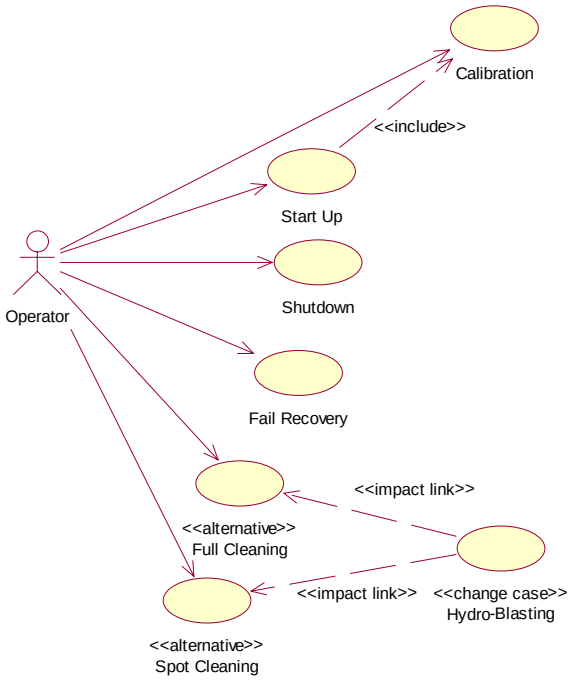

Figura 28 Casos de uso relacionados con la limpieza del casco

Se establecen relaciones de traza N:M entre características y casos de uso. Por ejemplo, la característica *Cleaning* (*Limpieza*) se relaciona con varios casos de uso (*Full Cleaning–Limpieza Completa*, *Spot Cleaning–Limpieza Mancha*, *Start Up–Inicio*, *Shutdown–Apagado*, *Calibration–Calibración*, y *Fail Recovery–Recuperación de fallos*) mientras que el caso de uso *Spot Cleaning–Limpieza Mancha* se relaciona con varias características (*Cleaning–Limpieza*, *Spot–Limpieza Mancha*, *Automatic–*

Automático, y *Teleoperated–Teleoperado*). La traza se establece –en cualquier sentido– entre casos de uso y características o bien, de modo más fino, entre pasos dentro de casos de uso y características. La traza se establece textualmente a partir de las plantillas de características (Tabla 34 y Tabla 35) y de casos de uso (que se explica posteriormente y se ejemplifica en la Tabla 36).

Con esta propuesta los puntos de variación de la línea de productos se especifican en el modelo de características, evitando así la sobrecarga del modelo de casos de uso con toda la complejidad asociada a la variabilidad de la línea de productos. Sin embargo, para hacer los diagramas de casos de uso más legibles, y siguiendo a Gomaa [122], los casos de uso opcionales y alternativos están etiquetados con los estereotipos «optional» y «alternative». De esta manera, los casos de uso de tipo opcional y alternativo se distinguen visualmente con facilidad, mientras que para encontrar los detalles de la variabilidad se debe acudir al modelo de características. Se puede decir, por tanto, que el modelo de casos de uso se ha estructurado por medio de relaciones de traza con el modelo de características.

Con esta propuesta el modelo de características se puede considerar como una interfaz de alto nivel de la línea de productos. Sin embargo, en una línea de productos existe una variabilidad que es intrínseca a los casos de uso: la que se refiere a las posibles variaciones en los pasos de los escenarios de interacción con los productos de la línea de productos, según se seleccionen unas características u otras. Esta variabilidad se debe capturar necesariamente en la plantilla de descripción del caso de uso.

Después de revisar distintas aproximaciones para contemplar la variabilidad de una línea de productos en los casos de uso [96, 123, 129, 155, 289], se ha decidido adaptar la aproximación de Eriksson *et al.* [96], que primero fue aplicada a una porción del sistema, la de *Cleaning* (ver Figura 28) y después al sistema completo. Se ha elegido esta propuesta porque coincide con nuestro enfoque de la especificación de la variabilidad en la línea de productos, descrito en párrafos anteriores, y porque propone el uso de las siguientes técnicas, que nos han parecido interesantes:

- *Casos de cambio* (*change cases*), en relación con el posible impacto en los casos de uso de la adopción de futuras extensiones previstas en el sistema, aunque todavía no disponibles. Los casos de cambio son casos de uso especiales, originalmente propuestos por Ecklund *et al.* [92], que capturan posibles futuras extensiones del sistema. Mediante relaciones de traza *impact link* se indica qué casos de uso pueden verse afectados por cada caso de cambio. El GCR planea extender su trabajo para desarrollar sistemas que posean más funcionalidad que la del EFTCoR, y por tanto ha considerado prometedor el uso de los casos de cambio. Por ejemplo, en la Figura 28 se aprecia un caso de cambio, *Hydro-Blasting (Limpieza basada en agua a presión)*, que implica un cambio en la técnica de limpieza utilizada, que hasta ahora ha sido *Grit-Blasting (Limpieza basada en granalla)*. Como se recoge en la Sección 3.3.1, Clements y Northrop [57] también incluyen los casos de cambio entre las prácticas recomendadas de IR para líneas de productos.
- El modelado de la variabilidad en la descripción de casos de uso, haciendo uso de las siguientes técnicas:

- Parámetros locales y globales en la descripción del caso de uso (en línea con Mannion *et al.* [198]).
- Una versión extendida de la descripción textual del flujo de eventos, siguiendo una plantilla adoptada del RUP SE (*Rational Unified Process for Systems Engineering*) [52], en la que con una notación especial se expresan los pasos de los escenarios donde puede haber variación. El proceso de modelado es el siguiente: (1) la descripción de los flujos de eventos se realiza considerando primero el sistema como una caja negra; (2) después se revisa la descripción y se va sustituyendo cada paso de caja negra en una serie de pasos de caja blanca, en los que se muestra la interacción de los distintos subsistemas para soportar cada paso de caja negra.

En la Tabla 36 se encuentra la descripción del caso de uso *Spot Cleaning (Limpieza Mancha)*. En la descripción textual del caso de uso hemos usado la plantilla de Gomaa [122], incluyendo en la parte de los escenarios la anteriormente citada descripción textual del flujo de eventos de caja negra propuesta por Eriksson *et al.* Los literales *alt*, *opt*, *loop* y *par* se utilizan para etiquetar determinados pasos del caso de uso. Por ejemplo, en la Tabla 36, dentro del campo *Descripción*, se observan pasos que son alternativos, pues utilizan el mismo número y están etiquetados con *alt.* Por ejemplo, un paso 3 está ligado con la característica *Teleoperated* y el otro paso 3 con *Automatic*, reflejando que la acción se puede realizar de forma teleoperada o automática. Los dos pasos 4 *alt* son análogos. En el campo Descripción de la Tabla 36 también quedan reflejados pasos que son opcionales, como el (5) *opt*.

En la Descripción de la Tabla 36 se ha utilizado una variable global ($MAX_TIME_START) y dos variables locales (@MAX_TIME_TOOL, @MAX_TIME_SAFE_STOP) para expresar en la columna *Requisitos no funcionales* (*Budget requirements*) los tiempos máximos de respuesta a determinadas acciones dentro del caso de uso. Esta columna *Requisitos no funcionales* se refiere a requisitos no funcionales que están relacionados con un paso determinado del caso de uso. Por ejemplo, @MAX_TIME_TOOL se refiere al tiempo máximo permitido para activar la herramienta de limpieza (paso 4). Finalmente, la columna *Traza Hacia* (*Trace to*) contiene los comandos que operacionalizan el caso de uso y las características relacionadas con el paso del caso de uso.

En línea con Gomaa [122], se ha experimentado que el proceso de construcción de los modelos de características y de casos de uso es concurrente y bidireccional: la identificación de características muestra algunas que son candidatas a ser desarrolladas mediante un caso de uso, y la identificación de casos de uso y su operacionalización con comandos de los STO proporciona realimentación sobre el modelo de características.

3.4.5 Un diccionario visual del dominio a través de un esquema conceptual

El complejo dominio de los STO era desconocido para el investigador, por lo que parecía recomendable comenzar el modelado realizando un ECD, para comprender y describir el vocabulario del dominio, es decir, los conceptos del espacio del problema y las relaciones entre ellos. Mientras que el diagrama de características aporta la visión

externa de un eventual comprador del sistema acerca de qué servicios tendrá disponibles y con qué niveles de calidad, el ECD sirve como una especie de *diccionario visual* del dominio que ayuda a comprender los conceptos del dominio y sus relaciones. Los requisitos especificados en las plantillas de características, de casos de uso o de plantillas de atributos de calidad tienen como "objetos directos" los conceptos de este modelo de información. En este punto se ha adoptado la visión de Larman [174] sobre el uso de un modelo de clases de UML para desarrollar lo que llama *modelo del dominio*, extendido con información sobre la dinámica de los tipos de eventos del dominio a través de un diagrama de estados.

Nombre del caso de uso: *Spot Cleaning* (*Limpieza Mancha*)

Tipo: ~~obligatorio/~~alternativo~~/opcional~~

Actores: *(actor primario y actores secundarios)* Operator *(primario)*

Resumen: *(objetivo del caso de uso, especificado en lenguaje natural)* Limpieza del casco del buque en un área específica del casco (*spot cleaning*). Puede ser realizada de forma teleoperada o automática.

Traza hacia (*Trace to*): *(plantillas de atributos de calidad y características trazadas desde el caso de uso)*

-Plantillas de atributos de calidad: Gestión de Comandos (*Command Management*), Mecanismos de Parada (*Stop Mechanisms*), Acceso al Interfaz de Usuario (*Access to the User Interface*)

-Características: *Cleaning* (*Limpieza*), *Spot* (*Limpieza Mancha*)

Precondiciones: El sistema está iniciado y calibrado

Postcondiciones: La superficie del casco seleccionada se ha limpiado

Cuestiones abiertas: -

Descripción:

Paso	Acción del actor	Sistema	Requisitos no funcionales	Traza hacia
1	El caso de uso comienza cuando el operador selecciona la opción de limpieza	El sistema se inicia para llevar a cabo la operación de limpieza	El tiempo máximo de respuesta es $MAX_TIME_START	
2 *loop*	El operador mueve el primario para alcanzar la zona afectada	El primario se mueve		
3 *loop alt*	El operador ejecuta comandos para poner la herramienta en la zona afectada, siguiendo las imágenes de la superficie del casco	El secundario se mueve		VP-FEAT Execution Mode: Teleoperated
3 *loop alt*	El sistema de visión ejecuta comandos de los sistemas de posicionamiento para situar la herramienta en la zona afectada	El secundario se mueve		VP-FEAT Execution Mode: Automatic
4 *alt*	El operador activa la herramienta de limpieza	La herramienta se activa para la limpieza	El tiempo máximo de respuesta es @MAX_TIME_TOOL	VP-FEAT Execution Mode: Teleoperated
4 *alt*	El sistema activa la herramienta de limpieza	La herramienta se activa para la limpieza	El tiempo máximo de respuesta es @MAX_TIME_TOOL	VP-FEAT Execution Mode: Automatic
5 *opt*	El operador selecciona la opción de parada segura	El sistema para de forma segura	El tiempo máximo de respuesta es @MAX_TIME_SAFE_STOP	FEAT Emergency Stop
6	El operador comprueba la calidad de la limpieza	Si es OK concluye el caso de uso, sino ir a 3		
*7	En cualquier momento, el operador selecciona la opción de cancelación	La limpieza para en ese punto		

Tabla 36 Descripción completa del caso de uso *Spot Cleaning* (*Limpieza Mancha*)

Como concluyen Lee *et al.* [182], en un dominio nuevo que no es todavía maduro, la estandarización de la terminología del dominio y la utilización de términos estándar durante el análisis puede acelerar el proceso de identificación de características. El análisis de la terminología estándar es una forma efectiva y eficiente de identificar las características del dominio. En el ECD se deben especificar las entidades propias del dominio que aparecen (1) en las plantillas de las características; (2) en los requisitos textuales; (3) en los casos de uso; y (4) en las plantillas de atributos de calidad. En este caso de estudio, parte de la funcionalidad de los STO fue descrita primero en el ECD e incluida posteriormente en el modelo de características. Sin embargo, no todas las características en el modelo de características están presentes en el esquema conceptual. Las características de bajo nivel que se identificaron durante la construcción del modelo de características (por ejemplo, ciertas características de implementación) y la mayor parte de las características de los niveles intermedios del árbol de características no fueron incluidas en modelo conceptual.

En la ejecución del caso de estudio el modelo conceptual fue "congelado" después de proporcionar comprensión del dominio, un vocabulario común y un punto de comienzo para construir el modelo de características. El modelo conceptual no fue utilizado después excepto para documentar gráficamente el conocimiento sobre el dominio.

En la Figura 29 se muestra una parte del modelo de clases del ECD, en el que quedan reflejadas las interacciones de la RDCU con otros subsistemas: Subsistema de Visión, Subsistema de Monitorización y Subsistema de Reciclaje. Estos subsistemas pertenecen en realidad al espacio de la solución, a la arquitectura del sistema, pero han sido incluidos en el ECD para proporcionar una comprensión mayor del dominio de los STO, dado que para el GCR son "fijos" en su descripción del dominio. Por cuestiones de legibilidad en la Figura 29 sólo se muestran algunos de los atributos del diagrama original.

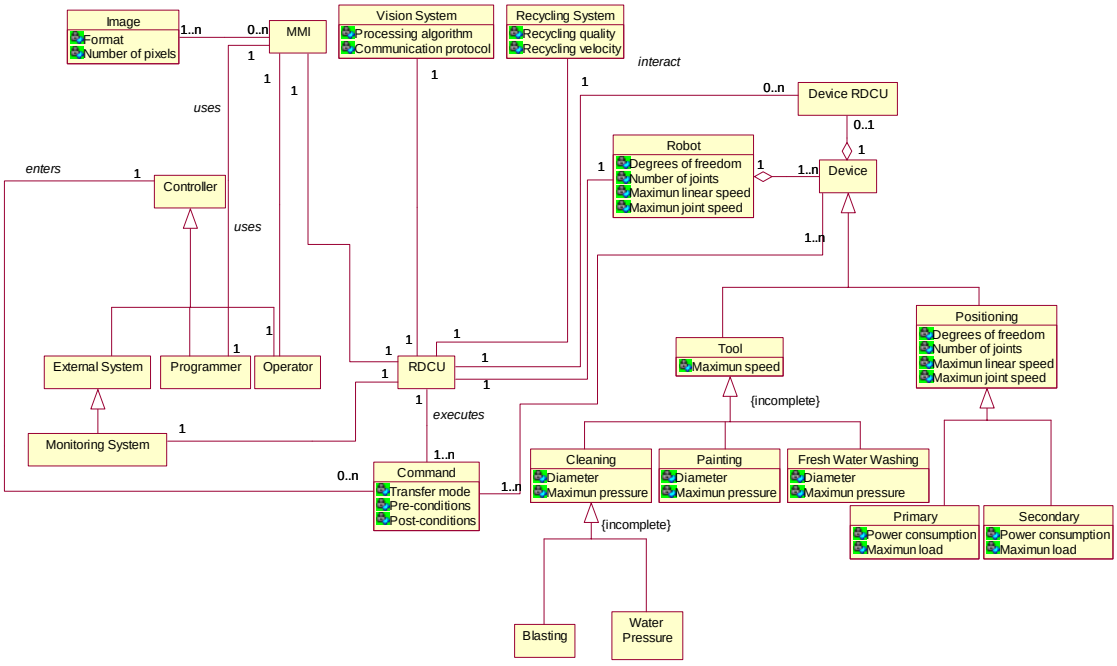

Figura 29 Un extracto de las entidades del ECD

El ECD se completa con información sobre los tipos de eventos del dominio que se describen a través de una máquina de estados UML (ver Figura 30), que muestra los modos de funcionamiento y los comandos habilitados de la RDCU. La máquina de estados especifica que, cuando el sistema comienza, permanece en un estado *Idle* (*Parado*) hasta que pasa a estado *Operational (En funcionamiento)*. Las transiciones entre los estados se producen por una señal *change_of_mode()*, con la excepción de la transición a *Safe stop* (*Parada segura*) que es el resultado de una alarma. Cuando la alarma es restablecida, el sistema retorna a *Operational* o a *Idle* dependiendo de la severidad de la alarma. El estado *Maintenance* (*Mantenimiento*) puede ser considerado como un estado de operaciones limitadas que permite que las operaciones de calibración, diagnóstico, configuración o aprendizaje sean realizadas en condiciones de seguridad a terceros (por ejemplo, se limita la velocidad de los ejes y la ejecución de comandos automáticos).

El investigador se ha planteado si hubiera sido interesante aumentar el grado de formalidad en la elaboración del ECD para llegar a la definición formal de una ontología del dominio haciendo uso de un lenguaje de definición de ontologías. Por ejemplo, para mantenernos dentro del estándar UML, se podría usar la propuesta de Guizzardi *et al.* [127], que definen un perfil de UML para realizar ontologías. El uso de un lenguaje de definición de ontologías introduce una sobrecarga en cuanto a esfuerzo y legibilidad, y por tanto el investigador ha determinado finalmente no utilizar una iniciativa de este tipo al no percibir una evidencia clara de los beneficios. En este caso de estudio se ha elegido y ha sido suficiente una aproximación más práctica como la de Larman [174].

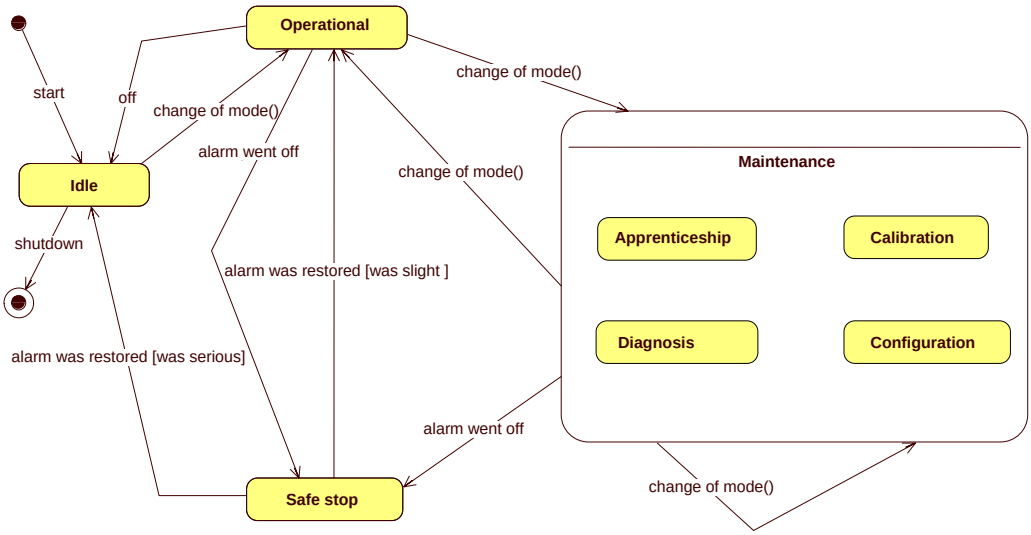

Figura 30 Diagrama de estados que describe los cambios de modo de la RDCU (*Robotic Devices Control Unit*)

3.4.6 Hacia la arquitectura y la implementación: atributos de calidad y comandos

En el diseño de una arquitectura genérica para los STO ciertas propiedades emergentes del sistema cobran especial importancia. Se trata de *atributos de calidad* que en este

caso de estudio han sido modelados siguiendo las recomendaciones de Bass *et al.* [36], dado que el GCR ya había trabajado con ellos en la especificación de su arquitectura de referencia [101]. Como se indica en la Sección 3.4.2, las *plantillas de atributos de calidad* (Tabla 37) incluyen una descripción de (1) los escenarios abstractos relacionados con los requisitos no funcionales que los sistemas de la línea de productos deben soportar; y (2) las respuestas que se deben producir. Según el marco para el análisis del dominio presentado en la Sección 3.4.2, esta primera parte de la plantilla de atributos se corresponde con el ECD. En las plantillas de atributos de calidad se proporcionan también posibles estrategias y mecanismos para gestionar dichos escenarios abstractos, además de relaciones de dependencia entre los atributos de calidad. En términos del marco para el análisis del dominio antes citado, esta segunda parte de la plantilla de atributo se corresponde con la EF. Las dependencias entre los atributos de calidad son más complejas que una dependencia inclusiva simple, pues incluyen relaciones de compromiso (*commitment relationships*) –como las identificadas por Chung *et al.* [72], que son básicamente *excluye*, *requiere*, *favorece* y *condiciona*–.

Las plantillas de atributos de calidad se pueden considerar como agrupaciones (*clusters*) que engloban requisitos, estrategias de diseño y mecanismos relacionados con una cuestión no funcional determinada. Según Bass *et al.* [36], los atributos de calidad se clasifican en las siguientes categorías o *aspectos*: (1) aspectos del negocio y objetivos de la organización; (2) aspectos de modificabilidad; (3) aspectos de rendimiento; (4) aspectos de seguridad; (5) aspectos de fiabilidad/disponibilidad; (6) aspectos de facilidad de uso; y (6) aspectos de interoperabilidad.

Descripción general.

Se describe en unas pocas líneas el propósito del atributo de calidad.

Escenarios abstractos.

Se definen los escenarios abstractos que ayuden a determinar el alcance. Los escenarios deben describirse en términos de los estímulos proporcionados al sistema y de las respuestas que debe proporcionar el mismo.

Estrategias y mecanismos arquitectónicos.

Definen los patrones y estilos arquitectónicos más apropiados para satisfacer el atributo de calidad. Su enumeración explícita facilitará la comparación entre las posibles alternativas y ayudará a una posterior evaluación de la alternativa escogida frente a las otras posibles.

Las estrategias se refieren a paradigmas y técnicas de programación generales, mientras que los mecanismos apuntan a soluciones más concretas. Por ejemplo, estrategias son el encapsulado, la separación de conceptos o proponer el uso de alguna técnica de asignación de recursos. Los mecanismos definen patrones arquitectónicos básicos (por ejemplo, el patrón de diseño Adapter) o complejos (por ejemplo, la arquitectura cliente/servidor) y describen técnicas concretas para implementar las estrategias arquitectónicas (planificación mediante prioridades fijas, ejecutivo cíclico, etc.).

Relación con otros atributos de calidad.

Se enumeran atributos de calidad con los que se relaciona y se explica brevemente el tipo de relación de compromiso (que se resumen en las siguientes: excluye, requiere, favorece, condiciona*).*

Relación con directrices de la arquitectura.

Se explica brevemente cómo se deriva el requisito de los objetivos del negocio, y del propósito y características de los sistemas.

Observaciones.

Cualquier información que se estime relevante, no incluida en los apartados anteriores.

Tabla 37 Plantilla de descripción de los atributos de calidad

En relación con la seguridad a terceros, en los STO es esencial controlar constantemente el funcionamiento del sistema, monitorizar la ejecución de comandos y proporcionar mecanismos de parada segura que son activados por el operador cuando se detectan situaciones peligrosas o cuando se detectan serios errores de funcionamiento. Los componentes críticos de los STO deben proporcionar interfaces a través de los cuales un servicio de diagnóstico o una tarea de supervisión puedan detectar fallos en su funcionamiento. Asimismo, la seguridad está estrechamente relacionada con otros atributos de calidad como el rendimiento y la disponibilidad del sistema. Por un lado, se deben garantizar las restricciones temporales para asegurar que la información disponible sobre el sistema refleja fielmente su estado, y por otro las tareas encargadas de monitorizar el funcionamiento del sistema suponen una sobrecarga del mismo. En general, los mecanismos y estrategias arquitectónicos que favorecen la disponibilidad son los mismos que favorecen la seguridad. Otras cuestiones de la seguridad son las relacionadas con el control de acceso al sistema y los permisos de ejecución de ciertas funciones del mismo, la modificación de parámetros de configuración, etc. Debe garantizarse el acceso a dichas funciones sólo a los operarios cualificados (a través de claves de acceso u otros mecanismos), precisamente para que no se produzcan empleos incorrectos o peligrosos del sistema.

La monitorización de comandos es necesaria para prevenir la ejecución de comandos no seguros y para asegurar que los comandos son ejecutados de acuerdo con la planificación prevista, como se muestra en la plantilla de atributos de calidad de la Tabla 38. En esta tabla los estímulos (*stimuli*) pueden ser vistos como las “causas” del escenario abstracto de la línea de productos, mientras que las respuestas (*responses*) pueden ser vistas como las “soluciones” propuestas. En la Tabla 38 la respuesta (a) se corresponde con el estímulo (a), “El sistema debe inhibir comandos que son incompatibles con la condición del sistema por razones de seguridad a terceros”. Por ejemplo, el sistema no se debe mover autónomamente si está en modo de mantenimiento. La misma correlación se puede encontrar entre el estímulo (b) y la respuesta (b), y así sucesivamente. El campo siguiente de *Estrategias y mecanismos de arquitectura* (*Architectural strategies and mechanisms*) especifica estrategias y mecanismos para manejar o resolver estos escenarios. Por ejemplo, cada estado del sistema se asocia con un número de comandos que pueden ser ejecutados en él. Como vemos, con las plantillas de atributos de calidad el modelado del dominio se solapa con el diseño arquitectónico al identificar las estrategias y mecanismos arquitectónicos más apropiados para conseguir un atributo de calidad determinado.

Resumiendo, en este caso de estudio los requisitos no funcionales se han especificado por medio de (1) *características de calidad* (*quality features*), que son las características en el modelo de características relacionadas con cuestiones no funcionales [161]; (2) plantillas de atributos de calidad, que se trazan a las características de calidad anteriores; y (3) casos de uso genéricos, en los cuales los requisitos no funcionales se especifican de dos maneras, mediante trazas a las plantillas de atributos de calidad (campo *Traza Hacia*) y en los pasos del caso de uso (campo *Requisitos no funcionales* de la plantilla de casos de uso). La utilización de plantillas de atributos de calidad para especificar las cuestiones no funcionales es estudiada con más profundidad en el ámbito de la tesis del doctorando del GIS Joaquín Lasheras.

Aspectos de seguridad a terceros (*Safety aspects*):
Gestión de comandos (*Command management*)

Descripción general:

Gestión de comandos para:

Prevenir la ejecución de comandos no seguros

Asegurar que los comandos son ejecutados de acuerdo con la planificación

Escenarios abstractos:

Estímulos:

(a) El sistema entra en una condición en la que no es seguro ejecutar ciertos comandos

(b) La ejecución del comando es compatible con el estado actual del sistema, pero no es posible indicar si es segura

(c) El comando no se ejecuta de acuerdo con la planificación

Respuestas:

(a) Deshabilitar comandos no compatibles con la condición del sistema

(b) Comprobar la viabilidad de un comando antes de que sea ejecutado (un comando es viable si el estado del sistema después de la ejecución es conocido y seguro)

(c) Monitorizar la ejecución de comandos comprobando que son llevados a cabo de acuerdo con la planificación (no existen discrepancias inaceptables entre las condiciones reales y esperadas, no hay retardos, etc.)

Estrategias y mecanismos arquitectónicos:

Gestión de condiciones y modelización de comandos:

- Cada condición se asocia con un número de comandos que pueden ser ejecutados en él
- Cada comando puede ser ejecutado por el sistema en ciertas condiciones

Separación de conceptos y encapsulación:

- Con cada condición, encapsular los comandos que pueden ser ejecutados en ella
- Con cada comando, encapsular las condiciones en las cuales puede ser ejecutada

Antes de la simulación de comandos

(...)

Relaciones con otros atributos de calidad:

Rendimiento: (...)

Modificabilidad: (...)

(...)

Observaciones:

(...)

La simulación de comandos en línea puede sobrecargar seriamente el sistema. Un esquema alternativo y bastante útil consiste en la utilización de datos desde una simulación *off-line* para generar las condiciones esperadas y compararlas periódicamente con las condiciones actuales

Tabla 38 Plantilla de atributos de calidad asociada a Gestión de Comandos (extracto)

La variabilidad en los atributos de calidad se expresa a través de los escenarios abstractos que están incluidos en las plantillas (mostrando alternativas) y a través de las relaciones con otros atributos de calidad (excluye, requiere, favorece y condiciona).

Los comandos de la familia de productos son *eventos funcionales* que se intercambian entre los subsistemas en orden a especificar qué operaciones requiere cada subsistema

de los otros. Por ejemplo, cuando el Subsistema de Visión detecta un área que hay que limpiar envía un comando *jog_motion(joint_id, angle, position, direction, velocity, acceleration)* a la RDCU (*Robotic Device Control Unit*) para forzar al robot a que se mueva al área deseada. Habitualmente los STO comparten los mismos comandos entre aplicaciones aunque su implementación puede ser diferente. En nuestra propuesta se puede acceder directamente a estos comandos desde (1) las características; (2) los requisitos textuales; y (3) los pasos en los casos de uso. Los comandos STO se describen en plantillas que constan de su significado, parámetros, tipo, restricciones y pre y postcondiciones. A modo de ejemplo en la Tabla 39 se presenta un extracto de la plantilla que define los parámetros principales de un comando *jog_motion*.

Comando	*Jog_motion()*
Significado	Imparte una orden para mover una articulación un ángulo o una posición (si es linear)
Parámetros requeridos	*joint_id, angle, position, direction, velocity, acceleration*
Parámetros opcionales	*Max. torque, acceleration profile*
Tipo	Asíncrono
Restricciones	Límites, fin de carrera (*end of stroke*)
Precondiciones	El robot está compuesto de *n* articulaciones que pueden ser manipuladas individualmente o de forma coordinada Una fuente de movimiento (*motion source*) ha sido establecida
Postcondiciones	La articulación *i* ha sido movida a una posición angular o linear con respecto a una fuente de movimiento
Retorno	Éxito / Fallo

Tabla 39 Especificación del comando *Jog motion* (extracto)

3.4.7 Modelo de procesos de SIRENspl

En este apartado se reflexiona sobre la definición de un modelo de procesos para SIRENspl. Esta cuestión no se puede reducir simplemente a pensar en cómo se podría interpretar el modelo de procesos de SIREN (mostrado en la Figura 6 de la página 74) en el ámbito del modelado de una línea de productos, pues en dicha figura el desarrollo *para* reutilización está simplificado y se ciñe exclusivamente a la tarea T1, *Creación del Catálogo de Requisitos Reutilizables*, mientras el resto de tareas conforman una espiral, la del desarrollo *con* reutilización. Realmente, durante la ejecución de la tarea T1 se produce también un desarrollo iterativo e incremental que en el caso de SIRENspl sería conveniente especificar con más detalle para definir el orden de desarrollo de los modelos involucrados en el marco de la propuesta.

Gomaa [122] muestra cómo el clásico modelo de procesos en espiral de Boehm se traduce en un modelo de doble espiral en el ámbito de las líneas de productos (Figura 31): (1) una primera espiral para el desarrollo de la línea de productos, que se relaciona con (2) una segunda espiral para el desarrollo de los productos individuales. Al igual que en el modelo en espiral clásico, el análisis de riesgos juega un papel fundamental en

cada iteración de ambas espirales y la planificación de los ciclos se va adaptando iterativamente.

La aplicación de SIRENspl en el caso de estudio de los STO ha sido guiada más por la elaboración de los modelos seleccionados que por un modelo de procesos previamente definido. Creemos que los modelos implicados se pueden usar en el marco de cualquier modelo de procesos. En este caso de estudio los distintos modelos se han construido de forma concurrente, con continuos solapamientos y retroalimentaciones. Por tanto, nuestra propuesta se encuentra cercana a la de Gomaa [122], en la cual se considera que el modelado de requisitos para líneas de productos software consiste de las tres actividades principales siguientes: (1) Ámbito de la línea de productos (*product line scoping*); (2) Modelado de casos de uso; y (3) Modelado de características. Como vemos, (2) y (3) se siguen de los modelos utilizados en el ámbito de los requisitos en este enfoque. Gomaa [122] muestra cómo su propuesta se puede integrar con modelos de procesos preexistentes, como el modelo en espiral y el modelo de procesos del Proceso Unificado.

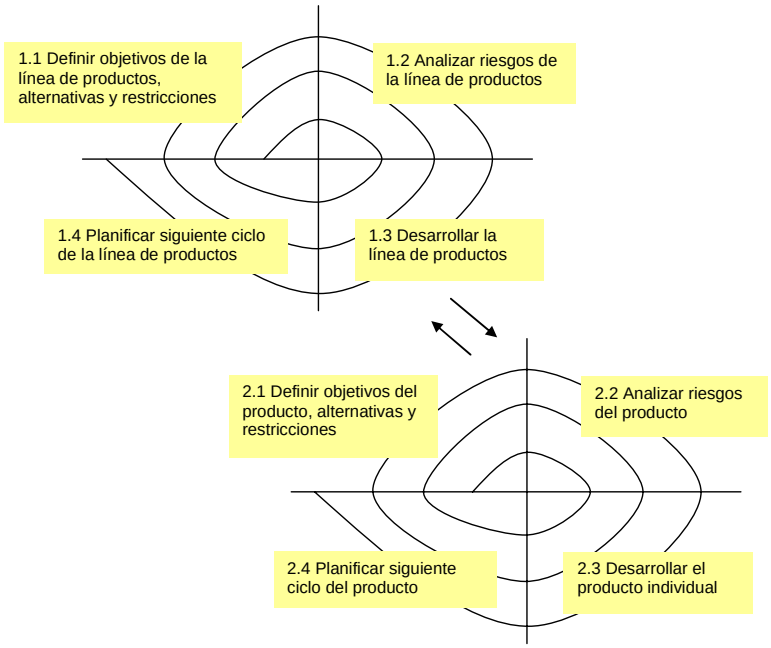

Figura 31 Modelo de procesos de doble espiral para líneas de productos (adoptado de Gomaa [122])

Cuando examinan el proceso de IR del dominio, Pohl *et al.* [249] añaden las siguientes tres actividades a las actividades clásicas de IR (presentadas, por ejemplo, en la Sección 2.5.1): (1) *Análisis de partes comunes* (*commonality analysis*), para identificar los requisitos comunes a toda la línea de productos; (2) *Análisis de partes variables* (*variability analysis*), para determinar de forma precisa los requisitos que difieren entre las aplicaciones; y (3) *Modelado de la variabilidad* (*variability modelling*), para modelar los puntos de variación, las variantes y sus relaciones. Se trata de tres actividades específicas de la IR del dominio que tienen por objetivo la definición precisa de la variabilidad, y que Pohl *et al.* [249] discuten en detalle. Nuestra propuesta para SIRENspl se encuentra más cercana a la de Gomaa [122] comentada anteriormente que a la de Pohl *et al.* [249]. Pensamos que las actividades específicas de la IR para líneas de productos mencionadas por Pohl *et al.* [249] serían entonces *transversales* a

las actividades de SIRENspl, al igual que creemos que ocurre en la propuesta de Gomaa [122].

El modelo de procesos de SIRENspl se define con base en las siguientes tareas: SPL1–*Establecimiento del ámbito de la línea de productos*; SPL2–*Modelado conceptual*; SPL3–*Modelado de características y requisitos asociados*; SPL4–*Modelado de casos de uso*; SPL5–*Especificación de plantillas de atributos de calidad*; y SPL6–*Especificación de comandos*. En la Figura 32 se muestra gráficamente este modelo de procesos con la notación de SPEM, utilizada en el Capítulo 2 para mostrar los modelos de proceso de SIREN y de SIRENgsd.

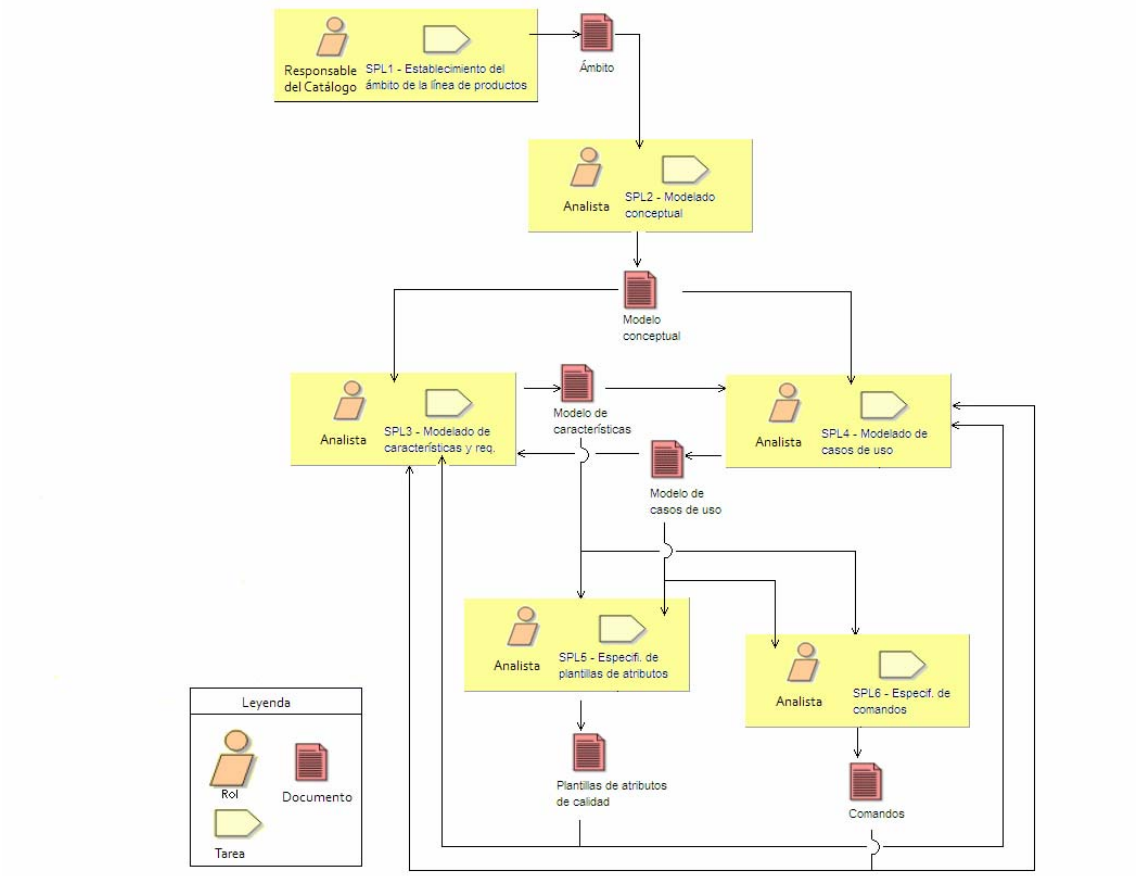

Figura 32 Modelo de procesos de SIRENspl con la notación SPEM

La tarea SPL1, *Establecimiento del ámbito de la línea de productos*, tiene por objetivo estimar la funcionalidad de la línea de productos, los límites y el tamaño de la misma, el grado de funcionalidad común con respecto a la funcionalidad total de la línea de productos y el grado de variabilidad. También tiene por objetivo realizar una identificación preliminar de los miembros potenciales de la línea de productos. En esta tarea se realiza un modelo conceptual y un modelo de características preliminar. Esta tarea se realizaría en el cuadrante 1.1 de la Figura 31.

El significado del resto de las tareas SPL2–SPL6 es bastante claro, pues cada tarea está centrada en un modelo del marco de análisis del dominio propuesto. Estas cinco tareas SPL2–SPL6 se ejecutarían iterativamente en el tercer cuadrante de la espiral para desarrollo de líneas de productos de la Figura 31. La ejecución de estas tareas no se

distribuye uniformemente entre las iteraciones de la espiral. En las primeras iteraciones se desarrollaría SPL2; en las siguientes iteraciones se desarrollarían SPL3 y SPL4, concurrentemente; finalmente SPL5 y SPL6 se ejecutarían en las últimas iteraciones. Evidentemente no tratamos de proponer un proceso *prescriptivo*, pues en cualquier momento se pueden producir solapamientos y retroalimentaciones entre las distintas tareas. Se puede observar cómo este modelo de procesos está centrado en la IR del dominio y no en la IR del producto.

3.4.8 Lecciones aprendidas

El dominio de los STO se definió después de construir un primer prototipo denominado GOYA [239]. Este prototipo constituyó el primer robot para mantenimiento de buques desarrollado por el GCR y permitió realizar un estudio exhaustivo del dominio de los STO y definir modelos iniciales de características y de casos de uso y plantillas de atributos de calidad. Los modelos de los STO fueron entonces instanciados en el proyecto EFTCoR para los productos que se muestran en la Tabla 40, que fueron construidos por el GCR. El conocimiento adquirido durante el desarrollo de estos sistemas EFTCoR ha enriquecido a su vez el modelo del dominio de los STO.

Identif. de producto	**Operación de limpieza y área del casco del buque**	**Sistema primario**	**Sistema secundario**	**Herramienta**
P1-SV	Limpieza mancha (*spotting*) (superficies verticales)	Torres verticales	Mesa XYZ	Cabeza limpiadora con confinamiento de residuos
P2-SB	Limpieza mancha (*spotting*) (bajos)	Grúa de tijera	Mesa XYZ	Cabeza limpiadora con confinamiento de residuos
P3-FV	Limpieza completa (*full blasting*) (superficies verticales)	Torres verticales	---------	Limpieza basada en turbinas
P4-FS	Limpieza completa (*full blasting*) (superficies curvas)	Torres verticales	---------	Cabeza limpiadora con confinamiento de residuos

Tabla 40 Productos EFTCoR para los cuales los modelos han sido instanciados

A pesar de que algunos miembros del GCR que participaban en el proyecto EFTCoR habían desarrollado previamente el prototipo GOYA, fue difícil recoger datos cuantitativos que fueran representativos para una comparación entre ambos proyectos porque el proyecto GOYA no había sido desarrollado teniendo en cuenta en la especificación de requisitos las técnicas, métodos y herramientas de ingeniería del software y no se habían recogido datos. En este contexto, sólo podemos proporcionar los datos generales reportados por el GCR en relación con el tiempo de desarrollo, las personas implicadas y el número de sistemas que fueron construidos en ambos proyectos (ver Tabla 41). En cualquier caso, creemos que el método de investigación cualitativo escogido, I-A, ha sido útil porque ha dirigido la investigación hacia objetivos prácticos y ha ayudado a la industria a aplicar en la práctica los resultados de la investigación.

Proyecto	Duración	Personal fijo	Personal auxiliar	Sistemas construidos
GOYA	2 años	8 personas	4 personas	1 prototipo
EFTCoR	3 años	12 personas	4 personas	4 sistemas

Tabla 41 Tiempos de desarrollo y sistemas desarrollados en GOYA y EFTCoR

En esta sección discutimos las principales lecciones aprendidas en este caso de estudio, que resumen la experiencia del personal del proyecto en el análisis del dominio de los STO. Estas lecciones aprendidas se organizan en grupos para facilitar su lectura. Las impresiones del GCR respecto a la adopción del modelo del dominio de los STO han sido recogidas y evaluadas a través de entrevistas, observaciones y discusiones de grupo. Además, para cuantificar sus experiencias y opiniones, el personal implicado en el desarrollo de los STO (diez personas) rellenó un cuestionario de 20 puntos.

Las características de FORM son útiles en la estructuración de un catálogo de requisitos reutilizables y aceleran la búsqueda de requisitos en el catálogo

Antes de esta experiencia, como se comenta en el Capítulo 2 de esta memoria de tesis doctoral, el investigador había trabajado aplicando SIREN en la reutilización de requisitos textuales en los dominios de protección de datos personales [285] y de seguridad [284]. El investigador era consciente por tanto de que un catálogo de requisitos reutilizables en lenguaje natural para un dominio amplio puede ser suficientemente preciso y correcto, pero en ocasiones difícil de manejar, al estar formado por largas listas de requisitos textuales que están ordenados en secciones en una jerarquía de documentos. Por ejemplo, a veces es difícil localizar los requisitos que están ligados con una determinada cuestión. En orden a evitar este problema, la estructura del catálogo de requisitos de los STO no se ha estructurado como una lista de requisitos, fundamentalmente textuales, estructurados en una jerarquía de documentos estándar, sino que las características (que son esencialmente características FORM) sirven de punto de partida para estructurar directamente el catálogo de requisitos. Con el uso de características se busca razonar sobre la definición de un producto de la línea de productos de forma más ágil que con el uso exclusivo de requisitos textuales, y al margen de la localización de tales requisitos en los documentos de requisitos. Así, durante la especificación de un producto concreto dentro de la línea de productos, se navega en el espacio del problema a través del *espacio de decisión* que viene dado por las características, seleccionando unas u otras, de forma más sencilla que recorrer listas de requisitos para seleccionar unos u otros. Además, durante la especificación inicial de requisitos es más cómodo realizar un análisis de características que no escribir requisitos textuales completos de calidad, que como establece el estándar IEEE 830-1998 [138], han de ser no ambiguos, completos, consistentes y verificables.

En relación con el GCR, casi todas las personas involucradas en el proyecto que habían tenido un conocimiento previo del dominio al haber trabajado ya en el prototipo GOYA encontraron el modelo de características útil y fácil de usar en la definición de un nuevo producto, y para definir las principales características de un nuevo producto rápidamente preferían la descripción gráfica de las características en lugar de la descripción textual de las mismas a través de sus plantillas. Para el 90% de los encuestados en el proyecto EFTCoR la representación gráfica era la más apropiada, pero la mayor parte de ellos también sugerían el uso de una herramienta software de

soporte para gestionar el modelo gráfico, que sería especialmente útil cuando el modelo fuera más grade.

Como Griss *et al.* [126], para gestionar la complejidad el GCR decidió que era interesante incluir de alguna manera vistas en el modelo de características, y así se incluyeron dos niveles de abstracción –capacidad e implementación–. Estos niveles han probado ser útiles para tratar con la complejidad del modelo al tiempo que simplifican los cuatro niveles de características de FORM. La experiencia en el proyecto EFTCoR refleja que el 80% de las personas involucradas en el desarrollo de los STO apreciaban positivamente el uso de dos vistas para separar capacidad e implementación.

Obviamente, las características no son la única vía para modelar los requisitos de una línea de productos. Para organizar una especificación de requisitos, herramientas CARE como IRqA [145] proponen el uso de unos *bloques* que de forma similar a las características permiten formar *clusters* de requisitos relacionados. Los bloques tienen propiedades que son heredadas por todos los requisitos que contienen. Existe un conjunto de bloques predefinidos, que incluyen los siguientes: *conceptos, funcionales, no funcionales, rendimiento, seguridad* y *servicios* (obsérvese como son similares a los principales conceptos que hemos usado en la descripción de los STO). Los bloques se pueden relacionar a través de las relaciones de UML de asociación y generalización. Los diagramas de bloques muestran gráficamente la especificación y facilitan su navegación, pero no se han elegido para modelar los STO debido a que los bloques tienen un tamaño de grano mayor que el de las características, y los diagramas de bloques no permiten la representación gráfica de la variabilidad. Además, se trata de una solución propietaria y la herramienta no es de libre distribución. Estas críticas se pueden trasladar a otras herramientas CARE como Borland Caliber-RM [6] que proporcionan diagramas de bloques similares.

Las características FORM son útiles para especificar la variabilidad en los requisitos

Podemos comparar de nuevo esta experiencia con el trabajo anterior del investigador en los dominios de protección personal de datos [285] y de seguridad [284]. En dichos trabajos no se desarrolló explícitamente ningún modelo de variabilidad. La variabilidad en los requisitos en lenguaje natural se expresó en tales trabajos por medio de (1) lenguaje natural en el texto de los requisitos; (2) requisitos parametrizados; y (3) trazas inclusivas y exclusivas entre requisitos. En consecuencia, en estas primeras experiencias la variabilidad se encuentra esparcida por toda la especificación de requisitos, dando lugar a los problemas enumerados en la Sección 3.3.2.4.

Sobre la base de la experiencia en el dominio de los STO, el investigador cree que el modelo de características contribuye a tener una especificación más clara de la variabilidad en la línea de productos que en las experiencias mencionadas en el párrafo anterior [284, 285]. Con las extensiones realizadas en el ámbito de este caso de estudio, el modelo de características queda con una expresividad similar a los modelos de variabilidad independientes introducidos en la Sección 3.3.2.3. El modelo de características juega así el papel de un *modelo de variabilidad* que abarca la variabilidad de la línea de productos que es relevante para un eventual comprador de un producto de la línea de productos: el modelo de características muestra la variabilidad gráficamente y extiende los constructores relacionados con la variabilidad que están presentes en las aproximaciones anteriores. La representación gráfica de los puntos de variación en el

modelo de características (*vp-features*), que extienden la notación de FORM, ha sido útil para hacer explícitas las decisiones que han de ser tomadas en la instanciación de la línea de productos. Sin embargo, la utilización de una notación más expresiva para denotar las cardinalidades ha sido útil verdaderamente sólo para mostrar nodos XOR y nodos OR.

Como se ha mostrado en la Sección 3.3.2.4, Pohl *et al.* [249] proponen el uso de un modelo de variabilidad ortogonal que resuelve ciertas limitaciones de los modelos de características propuestos inicialmente por FODA y FORM. Por un lado, pensamos que el uso de un modelo de variabilidad ortogonal cuando se va a utilizar un modelo de características para describir la línea de productos produce un cierto solapamiento en la definición de los puntos de variación de la línea de productos. Por otro lado, Pohl *et al.* [249] informan de dos debilidades en la especificación de la variabilidad en los modelos de características (Sección 3.3.2.5): (1) no se puede distinguir entre características alternativas que son comunes a todas las aplicaciones (y que son por tanto parte común de la línea de productos) y características alternativas que pueden ser seleccionadas por separado para un producto específico. En el dominio de los STO todas las características alternativas han de ser seleccionadas en tiempo de desarrollo, con lo cual esta debilidad del modelo no es relevante; y (2) el árbol de características no proporciona un mecanismo de agrupación que permita a un conjunto arbitrario de características ser asignado a una variante particular. La reestructuración del árbol de características no es siempre una alternativa viable, pues entonces se pierde la descomposición original de las características. No cuestionamos su aplicabilidad general, pero en el dominio de los STO no hemos necesitado estas *características paquete* (*package features*).

Durante la etapa de reflexión del caso de estudio presentado en este capítulo se ha determinado que la gestión de la variabilidad tal y como se realiza en este caso de estudio, con el uso exclusivo de modelos de características y de casos de uso, podría haber sido mejorada utilizando modelos de decisión como los propuestos en el ámbito de PuLSE (*Product Line Software Engineering*) [254, 269]. Con estos modelos se listan las alternativas para cada punto de variación y se especifican las consecuencias de escoger cada alternativa. Estos modelos de decisión son particularmente útiles en proyectos de gran escala. En un modelo de características grande establecer los distintos productos que componen la línea de productos no es tarea trivial. En este caso de estudio, sin embargo, el número de productos distintos no es grande y finalmente no se ha considerado necesario incluir la utilización de modelos de decisión.

Las trazas directas desde los requisitos a los comandos ayudan a reutilizar código

Para el GCR la traza a los comandos de los STO desde los requisitos de la línea de productos era importante. Con esta finalidad, las características, los casos de uso y las plantillas de atributos de calidad han sido operacionalizados con los comandos de los STO. Son estos comandos muy comunes tanto en la línea de productos de los STO como en otros robots, y sería extremadamente útil poder reutilizarlos. Por ejemplo, algunos de estos comandos son *move_primary_to(), move_secondary_to(), execute_sequence(), jog_x(),* y *lookfor_spot()*. Cuando se instancia un nuevo producto de la familia, a través del modelo de características podemos conocer qué capacidades e implementaciones necesita el nuevo producto –por ejemplo, la capacidad *Vision Navigation*–. Es muy útil tener una traza desde dicha característica al comando

lookfor_spot(), que incluye algoritmos que pueden ser reutilizados directamente. El desarrollador puede conocer así desde el principio qué comandos se necesitarán.

No todas las técnicas de esta propuesta han de ser usadas siempre, necesariamente, para modelar una nueva línea de productos desde el principio

El marco utilizado en este caso de estudio ha sido diseñado en el contexto de sistemas STO ya existentes. Como se afirma en la Sección 3.1, las arquitecturas de referencia habían sido desarrolladas en la línea de productos de los STO y las plantillas de atributos de calidad estaban disponibles. Se puede plantear si el marco de análisis del dominio planteado en este caso de estudio es el más adecuado para modelar una nueva línea de productos desde cero. Sabemos, como apunta por ejemplo Sommerville [276], que en ingeniería del software no existe una aproximación única, universal para todos los problemas. Por tanto, la propuesta debe ser abierta si se pretende que se pueda aplicar a otras situaciones.

- El investigador cree que si la línea de productos no está especialmente afectada por cuestiones de seguridad a terceros, puede ser adecuado especificar los requisitos no funcionales simplemente por medio de requisitos en lenguaje natural en lugar de mediante plantillas de atributos de calidad. Estas plantillas añaden más información pero como contrapartida son también mucho más difíciles de escribir. Las plantillas de atributos de calidad son especialmente útiles en sistemas en los cuales la seguridad a terceros es crítica, como en el caso de los STO.
- El investigador cree que sería interesante construir un modelo conceptual antes de modelar las características y los casos de uso sólo si se necesitara comprender un número importante de nuevos conceptos en el dominio y sus relaciones. En este caso de estudio, el modelo conceptual ha sido útil para asistir en el desarrollo del modelo de características. Sin embargo, el investigador ahora conoce el dominio bien y en una situación como esta tal vez sería mejor no dedicar tiempo a la realización del esquema conceptual, para concentrar todos los esfuerzos en el modelado de características y de casos de uso genéricos, que se podrían construir directamente.

Las principales impresiones que el GCR ha recopilado en relación con la utilidad de la propuesta están resumidas en la Tabla 42. Es de destacar que el personal con conocimiento previo del dominio es más entusiasta en relación con la propuesta presentada en este capítulo. El GCR indica que esto es probablemente debido a que una parte importante de él había trabajado en el prototipo GOYA y es consciente de la amplitud y complejidad del dominio de los STO y de las dificultades de desarrollar tales sistemas sin un enfoque disciplinado desde los requisitos.

3.4.9 Validez de los resultados

En esta sección se proporciona una visión crítica de la validez de los resultados obtenidos en este caso de estudio, especialmente del papel que ha jugado I-A. En primer lugar debemos decir que creemos que las debilidades de I-A surgen de interpretaciones incorrectas en su aplicación más que de su misma naturaleza. Baskerville y Wood-Harper [35] y después sólo Baskerville [32] proporcionan un profunda discusión sobre

los riesgos asociados con I-A y sobre cómo se pueden gestionar. Dos problemas mencionados que en ocasiones han aparecido en el presente caso de estudio son los siguientes:

- La ausencia de un método que permita a investigadores y desarrolladores utilizar y comprender I-A.
- La ausencia de un modelo de procesos de la investigación suficientemente detallado que muestre los pasos a seguir en I-A.

	Modelo de características	**Modelo de casos de uso**	**Plantillas de atributos de calidad**	**Reutilización del modelo del dominio**
Personal del GCR con conocimientos previos del dominio	El 83% cree que es más útil que el modelo de casos de uso para comprender el dominio y para definir nuevos productos	El 33% cree que debe ser extendido (más de un 30%) para nuevos desarrollos	El 33% cree que son difíciles de usar y que no ayudan a explicar el dominio	El 83% cree que debe ser reutilizado en desarrollos futuros (al menos en un 75%)
Personal del GCR sin conocimientos previos del dominio	El 50% cree que es más útil que el modelo de casos de uso para comprender el dominio y definir nuevos productos	El 75% cree que debe ser extendido (más de un 30%) para nuevos desarrollos	El 75% cree que son difíciles de usar y que no ayudan a explicar el dominio	El 50% cree que debe ser reutilizado en desarrollos futuros (al menos en un 75%)

Tabla 42 Principales conclusiones reportadas por el GCR a partir de los cuestionarios

Estos problemas ponen en peligro el rigor en el proceso de la investigación. Existe un riesgo en que investigadores y desarrolladores *olviden* el marco de la investigación, y que yerren en la definición de los pasos de planificación, acción, observación y reflexión. Debemos reconocer que en ocasiones este problema ha aparecido en este caso de estudio. Si este caso de estudio fuera a comenzar de nuevo, el investigador definiría más formalmente los objetivos de los ciclos de I-A y se diseñarían ciclos más cortos con objetivos más limitados. Se debe tener en cuenta que I-A se utilizaba por primera vez tanto por el investigador como por el GCR. I-A se ha mostrado como un método de investigación con el cual es posible establecer un marco sencillo e intuitivo para la investigación, aunque en la práctica el trabajo conjunto del investigador y del GCR ha procedido con continuas vueltas atrás y realimentaciones. En este caso de estudio, por ejemplo, no ha sido posible determinar los modelos que han sido necesarios para llevar a cabo el modelado (planificación) hasta que éste ha comenzado (acción).

Para evitar los problemas anteriores y para mejorar el rigor de la aplicación de I-A, Estay y Pastor [97] proponen dirigir la investigación a través de una perspectiva de gestión de proyectos, generando una estructura de proyecto que englobe los elementos principales de I-A. Para Estay y Pastor *investigación* y *proyecto* son conceptos equivalentes.

Un problema más en la aplicación de I-A se produce cuando el investigador adopta el papel de mero *consultor* en el proceso, un rol impuesto algunas veces por contrato. Cuando esto sucede, el investigador tiene una capacidad limitada para cambiar la práctica. En este caso de estudio, sin embargo, los investigadores y la dirección del

GCR estaban en condiciones de comunicarse a través de un "lenguaje de ingeniería del software" común, y el GCR no ha impuesto límites en los posibles cambios en la práctica.

Las conclusiones extraídas en este caso de estudio tendrían más valor si se hubieran recogido datos cuantitativos. La mejora en el proyecto EFTCoR en relación con el proyecto GOYA es evidente a partir de la Tabla 41 (tres productos más construidos en el EFTCoR con sólo cuatro personas más y un año más de trabajo), pero se debe tener en cuenta que los números en el proyecto EFTCoR son mejores no sólo debido al modelado del dominio de los STO, sino también debido a la experiencia previa de seis de los miembros del equipo de desarrollo, que habían trabajado previamente en GOYA. Así pues una limitación de la investigación realizada es que las mejoras en la calidad y la productividad postuladas no han sido medidas experimentalmente de forma cuantitativa para el ciclo de desarrollo de software, sino sólo de forma cualitativa.

3.4.10 Trabajos relacionados

Al margen de la breve descripción del estado del arte recogida en la Sección 3.3, en esta sección se recogen dos trabajos que han servido de especial inspiración a la hora de abordar este caso de estudio.

A partir de su experiencia en sistemas de control de motores de aviones, Lam *et al.* [171] identifican diez cuestiones clave cuya presencia es beneficiosa en una aproximación a la reutilización de requisitos. La experiencia obtenida en el caso de estudio que presentamos nos permite validar la mayor parte de los pasos propuestos por Lam *et al.*, que comentamos de forma muy breve a continuación: (1) es conveniente generalizar la especificación de requisitos para que sea reutilizable, pero para mejorar la reutilización también han de ser considerados los detalles (como hemos hecho a través de características opcionales, alternativas y parametrizadas, y los dos niveles de abstracción en el modelo de características); (2) es conveniente identificar familias de productos para maximizar la reutilización (la propuesta seguida en este capítulo); (3) es conveniente evaluar las técnicas de reutilización en términos de su impacto en el proceso (lo cual nos ha llevado a trabajar con aproximaciones sencillas y bien conocidas, en ocasiones dejando al margen aproximaciones más potentes); (4) es conveniente usar el conocimiento del dominio para organizar los artefactos reutilizables (como hacemos a través del modelo de características, el modelo de casos de uso y las plantillas de atributos de calidad); (5) es conveniente introducir relaciones de traza inclusiva entre requisitos, de la forma de "el requisito B es posible sólo si el requisito A se cumple" (lo cual hacemos principalmente a través de las relaciones de descomposición jerárquica en el modelo de características); (6) es conveniente hacer explícito el contexto de la reutilización para evitar el mal uso de la misma (para lo cual usamos las características externas y las relaciones de compromiso entre características); y (7) es conveniente evaluar el impacto de la reutilización de requisitos en el resto de procesos del desarrollo, como el diseño. No ponemos en duda la utilidad las tres cuestiones clave restantes mencionadas por Lam *et al.*: sin embargo, atestiguamos que en esta experiencia no han sido relevantes: (8) no ha sido necesario aplicar el método propuesto por Lam *et al.* para identificar los requisitos parametrizados, que han sido identificados de manera *ad hoc*; (9) no ha sido necesario identificar *partes de requisitos conectables* (*pluggable requirement parts*), que se podrían haber usado para mejorar la definición e instanciación de las partes

parametrizadas de los requisitos (realmente en SIREN ya se hace algo parecido cuando se identifican los posibles valores de los parámetros de los requisitos parametrizados); y finalmente (10) no se ha validado la afirmación de que "partes del proceso de IR son reutilizables también", ya que no se ha prestado una especial atención a la definición del proceso en sí: de hecho, el caso de estudio ha estado guiado por los productos definidos en la Sección 3.4.2 más que por un modelo de procesos determinado.

Como se ha reflejado en el párrafo anterior, para Lam *et al.* [171] una de las cuestiones clave con vistas a mejorar la reutilización es adoptar el paradigma de línea de productos. Clements y Northrop [57] describen las prácticas específicas que son interesantes en IR para líneas de productos, que se han puesto en práctica en este caso de estudio con los STO: (1) técnicas de análisis del dominio; (2) modelado de características; (3) modelado de casos de uso; (4) modelado de casos de cambio; y (5) trazabilidad desde los requisitos a los productos de trabajo asociados. La única cuestión mencionada por Clements y Northrop que no se ha llevado a la práctica en este caso de estudio ha sido la del modelado de las vistas de los interesados, ya que sólo hemos trabajado con un "suministrador de requisitos", el GCR, formado por los desarrolladores principales de la línea de productos, personas con una idea muy clara de los requisitos que se deben asociar a los distintos actores del sistema.

3.5 Herramienta de soporte: SirenSPLTool

El proyecto fin de carrera de Gustavo Caro [53] ha servido para desarrollar un prototipo básico de editor gráfico de soporte al modelado de los requisitos de líneas de productos mediante el enfoque de SIRENspl. El editor se ha acotado a los diagramas de características y de casos de uso genéricos con las extensiones presentadas en las secciones anteriores y soporta la trazabilidad entre requisitos textuales y características.

En un estudio de alternativas de diseño para implementar el editor se han evaluado varias herramientas MetaCASE (incluyendo MetaEdit+ [13]), *MOSKitt feature modeler* [15] y Eclipse EMF/GMF [94, 118]. Finalmente se ha escogido Eclipse por varias razones:

- Eclipse EMF/GMF ofrece acceso sencillo a modelos y metamodelos y automatiza la construcción de los editores gráficos asociados, si bien la documentación asociada actualmente es mejorable.
- Eclipse es un proyecto *open source* y su arquitectura basada en *plugins* hace sencillo reutilizar y añadir nuevas funcionalidades. Existe un vasto ecosistema de *plugins*, muchos de ellos libres, que pueden ser integrados para proporcionar y/o complementar la funcionalidad requerida.
- Por compatibilidad con los proyectos fin de carrera de Jorge Hernández [135] y Ana Pla [248], referenciados en el Capítulo 4 de esta memoria cuando se presenta la generación de requisitos en lenguaje natural a partir de modelos SIRENspl. En el ámbito de estos tres proyectos fin de carrera se han creado metamodelos para diagramas de características y para casos de uso genéricos, tanto a nivel conceptual (utilizando MOF, Capítulo 4) como de implementación

(utilizando la notación Ecore propia de Eclipse EMF, Capítulo 4). Estos metamodelos son así comunes a los tres proyectos, de manera que los modelos creados con el editor gráfico son compatibles con las herramientas de generación de requisitos en lenguaje natural presentadas en el Capítulo 4.

Para validar el editor gráfico se han utilizado retrospectivamente los modelos resultado del caso de estudio de los STO. A modo de ejemplo, en la Figura 33 vemos una parte del diagrama de características introducido, en la cual la característica *Stop* está seleccionada, de manera que sus propiedades se presentan en el cuadro de abajo. En la parte inferior derecha del editor se incluye un mapa global del diagrama de características. En la Figura 34, por otro lado, se observa cómo el caso de uso *SpotCleaning* se traza a las características relacionadas.

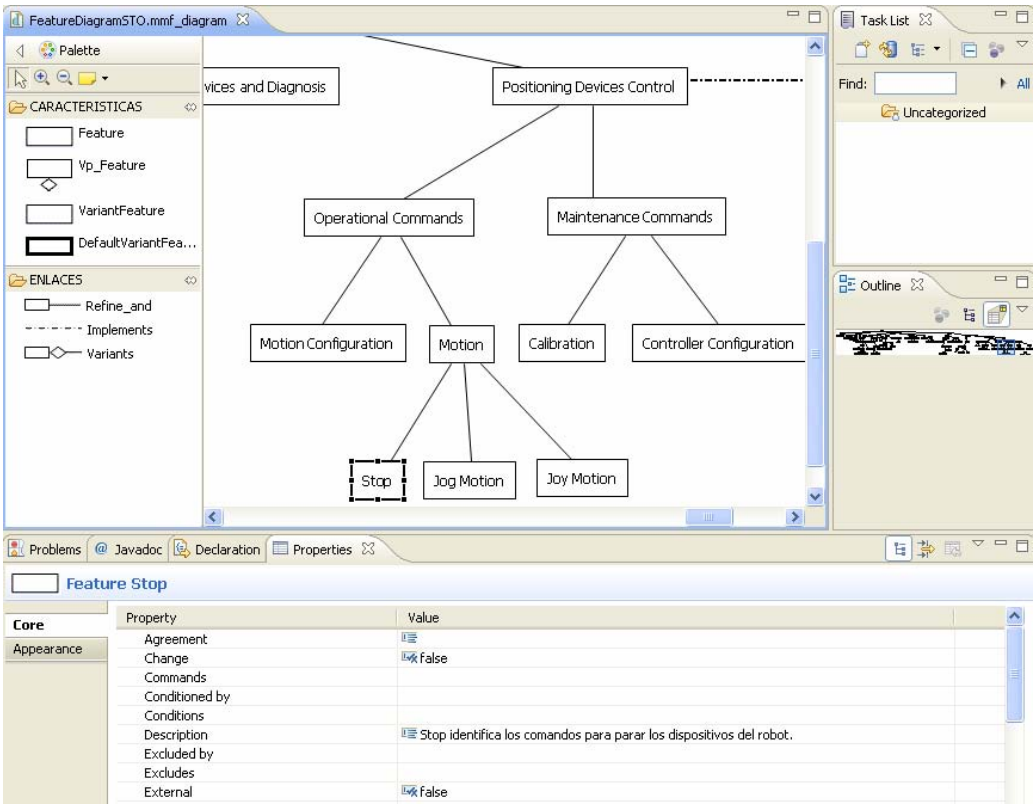

Figura 33 Extracto de una parte del diagrama de características

3.6 Conclusiones

En este capítulo hemos presentado el análisis de la línea de productos de los STO, con la intención de posibilitar la reutilización de requisitos en este dominio:

- Hemos presentado un modelo del dominio de los STO que se ha diseñado para incrementar el nivel de abstracción con el que los interesados pueden razonar sobre esta línea de productos.
- Hemos mostrado las principales lecciones aprendidas en este caso de estudio.

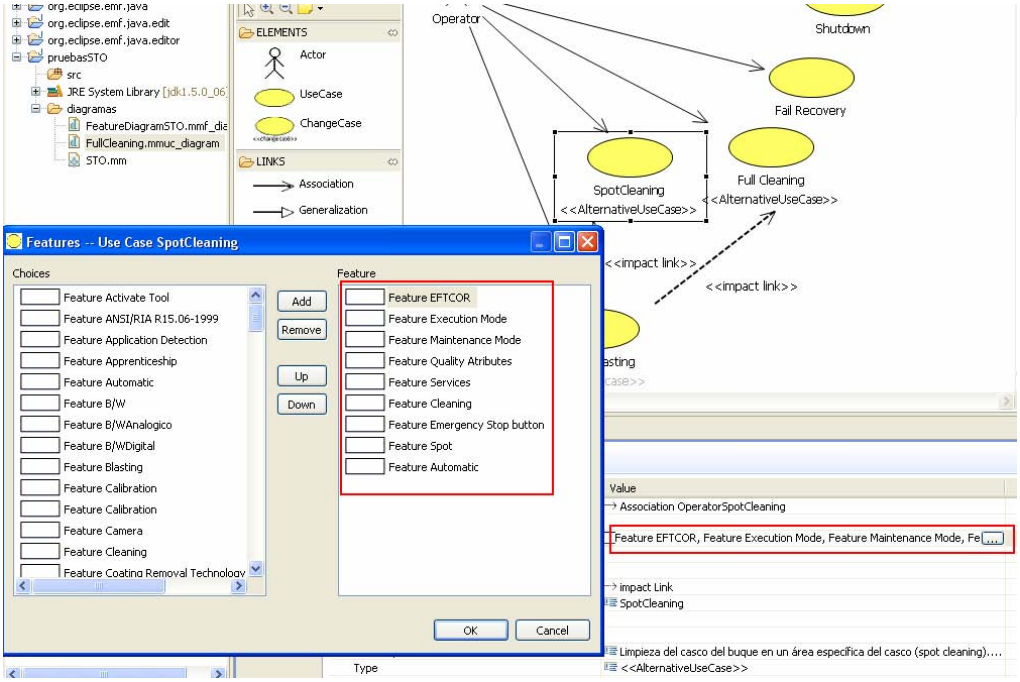

Figura 34 Traza entre las características y el caso de uso *SpotCleaning*

3.6.1 La aportación al estado del arte

En la medida de nuestro conocimiento, no existe una propuesta específica para afrontar la gestión de los requisitos de los STO. Las impresiones del GCR durante la adopción del modelo del dominio de los STO han sido recogidas y evaluadas mediante entrevistas, observaciones, discusiones de grupo y cuestionarios. Estas impresiones reflejan que el personal experimentado del GCR ha encontrado útil la adopción de técnicas de ingeniería del software. El GCR considera que el caso de estudio ha sido un éxito, teniendo en cuenta que el modelo de análisis que se ha desarrollado para la línea de productos le confiere un valor añadido en términos de su difusión entre clientes y desarrolladores, pues normalmente no es desarrollado en los sistemas teleoperados.

Creemos que el uso de técnicas de análisis del dominio y de requisitos tal y como se presenta en esta tesis doctoral puede ser extrapolado a otros sistemas teleoperados: los modelos de requisitos pueden ser diferentes, pero pensamos que la propuesta es válida para modelarlos, porque los escenarios de uso son similares y porque estos sistemas comparten un conjunto básico de comandos.

En términos de los campos utilizados en la síntesis de enfoques de reutilización de requisitos presentada en la Tabla 11 de la página 63, nuestro enfoque SIRENspl se resume como se recoge en la Tabla 43.

Llegados a este punto destacamos las siguientes aportaciones de SIRENspl:

- Para construir el modelo del dominio de los STO se ha realizado un análisis crítico del estado del arte en IR para líneas de productos, durante el cual se han seleccionado y enriquecido técnicas existentes, cuya aplicabilidad hemos

experimentado en este dominio. Para hacer la propuesta más accesible se han seleccionado técnicas bien conocidas en análisis del dominio, que se han enriquecido.

- El modelo de características FORM original se ha enriquecido con (1) la propuesta de las características de cambio, que se trazan a los casos de cambio; (2) la adopción de características externas; (3) la adopción de vp-features, que hacen explícitos los puntos de variación en el modelo de características; y (4) la adopción de una extensión para las cardinalidades de las vp-features.
- El modelo de características se presenta con una propuesta de dos niveles de abstracción, *capacidad* e *implementación*, que simplifican los cuatro niveles de abstracción originales de FORM.
- El modelo de casos de uso genéricos se traza al modelo de características, que en nuestro enfoque es el que aporta la especificación de la variabilidad, y se basa en los enfoques de Eriksson *et al.* [96] y de Gomaa [122].
- Todas las técnicas utilizadas se han ordenado en un marco sencillo adoptado del estudio de Olivé [232] sobre modelado conceptual (dividido en ECD, *Esquema Conceptual del Dominio* y EF, *Especificación Funcional*).

Gen.	Com.	Modelo de procesos	Técnicas y guías	Organización de requisitos	Herramienta	Validación
■	■	Sí, modelo de procesos iterativo sencillo (principalmente enfoque *para* reutilización)	Modelo de características FORM extendido, modelo de casos de uso genéricos, modelo conceptual, plantillas de atributos de calidad	Catálogos de requisitos por dominios de líneas de productos; MM de requisitos explícito utilizando MOF	SirenSPLTool, realizada mediante Eclipse EMF/GMF	(CEI) STO
Leyenda: Enfoque: *Gen*. Basado en generación; *Com*. Basado en composición; LDP: Línea De Productos; MM: Metamodelo; OO: Orientado a Objetos; CEI: Caso de Estudio Industrial / CEA: Caso de Estudio Académico						

Tabla 43 SIRENspl en términos de la síntesis de enfoques de reutilización de requisitos de la Tabla 11

3.6.2 Conclusiones adicionales y vías futuras

En esta sección podrían estar incluidas las cuestiones presentadas en la Sección 3.4.8, "Lecciones aprendidas". Además, podemos comentar que en el paradigma de LPS existe un consenso sobre la necesidad actual de prestar más atención al proceso de derivación del producto [159], en definitiva a la IR del producto o de la aplicación. Nuestra propuesta también se centra en la IR del dominio y no trata específicamente la IR del producto, de modo que una vía de trabajo futuro podría centrarse en la definición de este proceso.

En nuestra opinión, la propuesta aplicada en este caso de estudio podría haber sido más útil si durante el desarrollo del mismo se hubiera dispuesto de una herramienta para generar automáticamente la documentación de requisitos a partir de los modelos de análisis. La idea sería definir y automatizar los mecanismos para transformar los modelos de análisis del dominio en requisitos textuales, que serían etiquetados con atributos como por ejemplo *prioridad* o *grado de cumplimiento*. Una herramienta de soporte de líneas de productos, GEARS [11], ya permite la generación de la documentación de requisitos a partir de las características seleccionadas durante la derivación del producto. La documentación que GEARS genera es una instantánea final de la derivación de un producto en la línea de productos, en el sentido de que el documento de requisitos no puede evolucionar junto con los modelos de la línea de

productos software. Pensamos que sería interesante ir un paso más allá, y proporcionar dos *vistas* sincronizadas de la línea de productos software: la *vista modelos* y la *vista textual*. En un desarrollo iterativo e incremental, la vista textual simplificaría la gestión de los requisitos del proyecto y podría ayudar al cliente a conocer el estado del proyecto y a formalizar los términos del contrato. De esta idea surge la tercera línea de trabajo recogida en esta tesis doctoral, la generación de especificaciones de requisitos textuales a partir de modelos de ingeniería del software, que se desarrolla en el ámbito de SIRENspl en el Capítulo 4 de esta memoria de tesis doctoral.

4 Integración de modelos y requisitos textuales en SIRENspl

4.1 Introducción

Una manera tradicional de documentar los requisitos de un sistema son las listas de requisitos escritos con un *estilo contractual* (en inglés típicamente en la forma *The system shall…*). En un sistema complejo estas listas, que son a veces denominadas un tanto despectivamente *listas de la compra*, pueden abarcar cientos de páginas. Como técnica de especificación de un sistema o de una porción de software, estas listas adolecen de un conjunto de problemas que nos parecen consensuados en la comunidad de ingeniería del software (ver por ejemplo Larman [174]) y que resumimos a continuación:

- Estas listas de requisitos textuales pueden abarcar cientos de páginas, con lo cual es muy difícil leer tales documentos como un todo y obtener una visión completa y coherente del sistema. Cuanta más gente lee los requisitos, más visiones distintas del sistema se obtienen.
- Estas listas crean a veces un falso sentido de comprensión mutua entre los interesados en el proyecto (*stakeholders*) y los desarrolladores. Estas listas, al estar escritas con un estilo contractual, proporcionan a los interesados un falso sentido de seguridad sobre los resultados que deben proporcionar los desarrolladores. Sin embargo, puede haber requisitos cruciales que son identificados más tarde en el proceso, y los desarrolladores pueden usar entonces los nuevos requisitos para renegociar los términos y condiciones del contrato a su favor.
- Estas listas de requisitos no son de ayuda en el diseño del sistema.
- Estas listas de requisitos proporcionan *poco contexto*:
    - Es muy difícil ver cómo los requisitos se ajustan unos con otros.
    - Es difícil identificar cuáles son los requisitos más importantes.
    - Es extremadamente difícil estar seguro de que se tienen la mayoría de los requisitos del sistema.

Con nuestra experiencia en SIREN en el uso de los catálogos de seguridad y de protección de datos personales (Capítulo 2), podemos corroborar algunos de los problemas asociados al uso de estas listas (Sección 2.11). No obstante, estas listas de requisitos también tienen propiedades positivas. Creemos que están consensuados los siguientes beneficios:

- Estas listas pueden servir como listas de chequeo de requisitos, que pueden ser de utilidad en las pruebas del software, del sistema o de aceptación.
- Estas listas proporcionan un contrato entre los promotores del proyecto y los desarrolladores, un *punto de firma* (como discutimos en la Sección 2.2.3.7).
- Estas listas pueden proporcionar una descripción de alto nivel en un sistema de tamaño grande.
- Estas listas facilitan la gestión de los requisitos del proyecto, pues los requisitos se han identificado explícitamente, y es factible registrar el número de requisitos implementados y por implementar, y el estado de cada uno de los requisitos.

Como comentamos en la Sección 2.2.3.4, a la hora de especificar un sistema o una porción de software, en ingeniería del software es común la utilización de notaciones de modelado que originalmente fueron concebidas para el diseño del sistema más que propiamente para un análisis ajeno a cualquier aspecto de diseño: por ejemplo, se suelen utilizar diagramas de clase UML o entidad/relación en el modelado de los datos; diagramas de flujo de datos o diagramas de actividades UML en el modelado funcional; diagramas de estados en el modelado dinámico o de comportamiento. En la Sección 1.1.4 recogemos distintas referencias que postulan el interés de la interconexión de los modelos de requisitos del sistema o del software, y que señalan lo conveniente del uso combinado de modelos de ingeniería del software y de requisitos textuales en lenguaje natural. En concreto, para Davis [82] el uso conjunto de modelos de ingeniería del software y de listas de requisitos en lenguaje natural constituye una de las buenas prácticas para mejorar el proceso de Ingeniería de Requisitos (IR), con el objetivo de conseguir aunar los beneficios de ambas técnicas de especificación. Por un lado las técnicas de modelado suelen tener una representación diagramática y son expresivas, concisas y facilitan a los desarrolladores la especificación y comprensión de los aspectos comunes y variables del desarrollo; por otro lado, las listas de requisitos textuales, preferentemente en lenguaje natural, facilitan el acceso de los clientes a la especificación, pueden servir como contrato entre clientes y desarrolladores y ayudan a la gestión de los requisitos del proyecto, clarificando el tamaño del mismo y el estado actual de desarrollo de los requisitos.

En este contexto, y a partir de lo comentado en la Sección 1.1.4 y en las conclusiones del Capítulo 3 (Sección 3.6), en esta tesis doctoral nos planteamos combinar las ventajas del uso de técnicas de modelado de análisis del dominio –según la propuesta de SIRENspl, Capítulo 3– y de listas de requisitos textuales –según la propuesta de SIREN, Capítulo 2–. La propuesta de Davis no contempla la generación automática o semiautomática de requisitos a partir de modelos de ingeniería del software: el uso habitual en ingeniería del software ha sido la escritura manual de dos especificaciones independientes, el catálogo de requisitos en lenguaje natural por un lado y los modelos de ingeniería del software por otro, que se deben mantener consistentes de forma manual. Aunque en la práctica modelos y requisitos textuales se desarrollan concurrentemente, lo habitual es que la especificación de requisitos en lenguaje natural se desarrolle como base a partir de la cual se derivan los modelos del sistema o del software. Existen trabajos académicos que han estudiado la generación de modelos a partir de especificaciones de requisitos, si bien en la medida de nuestro conocimiento no han tenido un impacto apreciable en la industria. Nuestra intención es ir un paso más allá de la propuesta de Davis, y estudiar la generación automática o fuertemente asistida de requisitos textuales candidatos a partir de los modelos del dominio utilizados en SIRENspl. Como vemos, se trata de un cambio de filosofía: generar requisitos a partir de modelos y no al contrario, como habitualmente se ha realizado de forma manual en la práctica industrial. Los beneficios que postulamos que se pueden obtener con esta aproximación se exponen en la Sección 1.1.4: básicamente perseguimos facilitar el uso combinado de modelos de ingeniería del software y de requisitos textuales, (1) facilitando la escritura de la especificación de requisitos textuales, que muchas veces es una tarea tediosa, pues reproduce parte de la información ya especificada en los modelos del sistema o del dominio; (2) mejorando la completitud de la especificación de requisitos textuales; y (3) ayudando a establecer trazas entre los modelos de ingeniería del software y los requisitos textuales.

Es necesario señalar que aspiramos a generar requisitos textuales *candidatos*, que serán aceptados o no por el ingeniero de requisitos durante la especificación del sistema. Para el ingeniero de requisitos, como para los clientes y usuarios, es más sencillo aceptar, modificar o

rechazar la propuesta de un requisito que identificar, consensuar y especificar completamente desde cero un nuevo requisito no ambiguo, completo, consistente y verificable. Decimos que pretendemos "generar" requisitos textuales, aunque tal vez sería más apropiado decir que aspiramos a "integrar" información ya presente en los modelos de partida en una nueva forma, un modelo textual. Lo ideal sería que los requisitos textuales, o al menos una parte significativa de ellos, se generaran de forma automática e iterativa, durante todo el proyecto. Sin embargo, no aspiramos a generar la totalidad de requisitos textuales: incluso en un dominio *cerrado* como el de una línea de productos, la instanciación de un nuevo producto de la línea puede contener unos *deltas*, es decir, unos requisitos propios del producto instanciado, que no pertenecen propiamente a la línea de productos. Lo ideal sería poder mantener dos vistas de la especificación del sistema, cada una con distintos usuarios: (1) una *vista textual*, accesible a todos los interesados, incluidos aquellos sin conocimientos técnicos, que fuera la base del contrato del desarrollo, y que ayudara a clarificar el estado del proyecto y el estado del desarrollo de cada uno de los requisitos; y (2) una *vista modelos*, que ayudara a los desarrolladores a conseguir una visión precisa del desarrollo y que sirviera como base para el diseño. Pueden aparecer entonces problemas de consistencia entre ambas vistas, especialmente en el contexto de un desarrollo iterativo e incremental. Por ejemplo, el valor de un atributo de un requisito generado automáticamente podría ser cambiado por el ingeniero de requisitos, y ese cambio se debería mantener en sucesivas iteraciones, no debería ser sobreescrito por sucesivas generaciones.

En este capítulo comenzamos desarrollando una revisión sistemática de la literatura sobre la generación de especificaciones de requisitos textuales a partir de modelos de ingeniería del software (siguiendo un artículo de Nicolás y Toval [229], Sección 4.2). Esta revisión se ha llevado a cabo con el fin de conocer si se han propuesto en la literatura ideas similares a las que reflejamos en este apartado y si existe evidencia sobre lo prometedor (o no) de estas propuestas. Como resultado de esta revisión sistemática concluimos que, aunque no se trata de una línea de investigación que haya recibido mucha atención hasta la fecha, existen propuestas recientes en esta línea por parte de autores de reconocido prestigio que confirman el interés de trabajar en este campo. Además, creemos que las líneas de productos proporcionan un dominio bastante *cerrado* en el que es factible la generación de una parte importante de la especificación del producto instanciado y por supuesto de la definición de la propia línea de productos. Así, en la Sección 4.3 detallamos la propuesta de generación de requisitos textuales SIREN a partir de modelos de análisis del dominio de SIRENspl –lo que denominamos el *aplanamiento* de los modelos de análisis–. Para ello incluimos un estudio exhaustivo de los dominios de partida y de destino de la generación de requisitos. Estas transformaciones dan lugar a requisitos de dos tipos: (1) requisitos *directos*, que *transcriben* lo ya especificado en el modelo; y (2) requisitos *literarios*, que como veremos *parafrasean* los modelos de análisis. En este contexto, en la Sección 4.4 se muestra una implementación y validación de esta propuesta de aplanamiento, a partir de los proyectos fin de carrera de Jorge Hernández [135] y Ana Pla [248]. Esta implementación está basada en MDA (*Model Driven Architecture*) y en particular en el lenguaje estándar del OMG QVT (*Query/View/Transformation*) [235]. Mediante esta implementación, para mostrar la viabilidad de la propuesta hacemos una aplicación retrospectiva del aplanamiento en los modelos de análisis del dominio para los sistemas teleoperados para mantenimiento de cascos de buques, descritos en el Capítulo 3. Finalmente, la Sección 4.5 recoge las conclusiones de este capítulo.

4.2 Generación de especificaciones de requisitos textuales a partir de modelos de ingeniería del software: revisión sistemática

4.2.1 Diseño de una revisión sistemática de la literatura

Como se comenta en la Sección 4.1, se ha planteado la realización de una revisión sistemática de la literatura (RSL) con el objetivo de conocer el estado del arte en cuanto a la generación de requisitos textuales (especialmente en lenguaje natural, aunque también consideraremos aquellos escritos en cualquier notación formal) a partir de modelos de ingeniería del software, y en particular de modelos del dominio en líneas de productos. En esta Sección 4.2.1 se resume el protocolo de la RSL. En la Sección 4.2.1.1 se establece con precisión el ámbito o alcance de la RSL. En la Sección 4.2.1.2 se listan las cuestiones que guían la revisión. En la Sección 4.2.1.3 se realiza la planificación del proceso de búsqueda. En la Sección 4.2.1.4 se definen los criterios de inclusión y exclusión. En la Sección 4.2.1.5 se indica cómo se planea evaluar la calidad en la RSL. En la Sección 4.2.1.6 se muestran los datos recogidos en los estudios seleccionados en la RSL. Finalmente, en la Sección 4.2.1.7 se presenta un análisis de los datos recogidos. La RSL ha sido diseñada y ejecutada por este doctorando, mientras que el tutor de esta tesis doctoral ha revisado el protocolo, los trabajos incluidos y excluidos, y ha discutido los resultados de la revisión con este doctorando.

4.2.1.1 Ámbito de la revisión

El objetivo general de la RSL ha sido establecido en la Sección 4.1, pero requiere un refinamiento para que el ámbito de la revisión quede definido con precisión: en esta RSL queremos revisar las correspondencias *m2rs*: MOD → REQ y *m2rd*: MOD → REQDOC presentes en la literatura, donde:

- *MOD* es el conjunto de todos los *modelos* utilizados en el desarrollo de software que proporcionan una representación gráfica, diagramática, de la especificación de un sistema o de una porción de software, como por ejemplo un diagrama de clases UML, un modelo de contexto SD de i*, un modelo de características o incluso un modelo de navegación de la interfaz gráfica de usuario –*GUI*, *Graphical User Interface*–. Un modelo se construye por medio de una *técnica*. En términos de la arquitectura de cuatro capas de MOF (*Meta Object Facility*) [238], MOD consiste de todos los modelos de Nivel M1 que son susceptibles de representación gráfica.
- *REQ* es un conjunto que consiste de todos los conjuntos de *requisitos textuales*. En resumen un *requisito* es una condición o capacidad que debe ser satisfecha o poseída por un sistema o porción de software (ver Sección 1.1.1). Un *requisito textual* es un requisito definido (1) como una sentencia de texto; (2) como una sentencia de texto con atributos, que incluyen relaciones de traza (por ejemplo, el modelo de requisitos de IEEE 830 [138]); o (3) como una plantilla rellena con texto (como por ejemplo en

VOLERE [265]). Las opciones (2) y (3) son claramente equivalentes. Los requisitos textuales pueden ser especificados por medio de lenguaje natural o por medio de lenguaje formal. Un ejemplo de lenguaje natural son las tradicionales declaraciones textuales del tipo "El sistema deberá (…)", que en inglés se denominan con frecuencia *shall-statements*. Un ejemplo de lenguaje formal es la notación SCR [83].

- *REQDOC* es el conjunto de todos los *documentos de requisitos* del sistema y/o del software, que con carácter habitual combinan descripciones narrativas, requisitos textuales y modelos gráficos.
- *m2rs* es una correspondencia que relaciona un modelo del sistema y/o del software *m* con un conjunto de requisitos *rs*.
- *m2rd* es una correspondencia que relaciona un modelo del sistema y/o del software *m* con un documento de requisitos *rd*.

En relación con el conjunto REQ, como se ha explicado en la Sección 4.1, estamos interesados especialmente en requisitos textuales escritos en lenguaje natural, pero los lenguajes formales juegan un papel importante en ciertos dominios y por tanto los incluimos en el ámbito de la RSL.

En relación con *m2rs* y *m2rd* estamos especialmente interesados en aquellos enfoques que son adecuados para su automatización y que permiten que estas correspondencias se realicen de manera automática o fuertemente asistida. Los enfoques que describen la creación de un documento de requisitos a partir de modelos se incluyen también en el ámbito de la RSL, aunque no sean susceptibles de ser automatizados, sino que solamente se formulen con la intención de guiar a un ingeniero de requisitos que escribe un documento de requisitos a partir de modelos de ingeniería del software. Nótese que *m2rs* y *m2rd* se definen como correspondencias y no como aplicaciones, de manera que un elemento en el dominio MOD puede tener más de una imagen.

4.2.1.2 Cuestiones de la investigación

Las cuestiones que se tratan de responder en esta RSL son las siguientes:

> RQ1. ¿Qué valor se otorga en la literatura a la generación de especificaciones de requisitos (requisitos textuales y documentos de requisitos) a partir de modelos de ingeniería de software?
>
> RQ2. ¿Qué técnicas han sido consideradas en este campo? (Nos referimos a las técnicas utilizadas para construir (1) los modelos de ingeniería del software iniciales; (2) los correspondientes requisitos textuales y/o documentos de requisitos; y (3) los procedimientos de transformación.)

4.2.1.3 Proceso de búsqueda

Para realizar la RSL se han seleccionado las siguientes fuentes:

- IEEE Digital Library (www.computer.org/portal/site/csdl/index.jsp)
- ACM Digital Library (portal.acm.org)
- Science@Direct (www.sciencedirect.com)

- MetaPress (Kluwer+Springer) (www.metapress.com)
- Wiley InterScience (www.interscience.wiley.com)
- Google Scholar (scholar.google.com)

A partir de las fuentes anteriores se indexan las principales revistas y eventos de la comunidad de ingeniería del software, en particular las relacionadas con la IR (por ejemplo, revistas como *Requirements Engineering Journal* y conferencias y talleres como *RE, ICRE, REFSQ, SREIS, AWRE* y *WER*). *Google Scholar* ha sido incluido en el anterior listado de fuentes con el fin de completar el conjunto de conferencias y talleres y para buscar literatura gris (como *white papers* e informes técnicos).

Todas las fuentes consideradas disponen de máquinas de búsqueda basadas en palabras clave. La cadena de búsqueda definida para esta RSL es la siguiente: ("from" OR "generation" OR "generating" OR "combination" OR "combining" OR "derivation" OR "deriving" OR "integration" OR "integrating") AND ("models" OR "specifications" OR "scenarios" OR "use cases" OR "features" OR "stories") AND ("documentation" OR "documents" OR "requirements").

4.2.1.4 Criterios de inclusión y exclusión

La aplicación tentativa de la cadena de búsqueda anterior ha mostrado que en muchos casos es suficiente con leer el título de las contribuciones para determinar su no selección como candidatos en la RSL, pues los términos de la consulta son muy comunes en la literatura y dan lugar a muchos trabajos que claramente no entran dentro del ámbito de la RSL. Cuando el título no es suficiente para determinar la inclusión del trabajo como candidato, se lee el resumen y, si es necesario, la introducción o incluso el trabajo completo.

En relación con los criterios de exclusión debemos señalar que las contribuciones candidatas que presentan herramientas de gestión de requisitos (*herramientas CARE, Computer-Aided Requirements Engineering,* o *RMDB, Requirements Management Data Base,* ver Sección 2.2.5) que no presentan procedimientos específicos para combinar modelos y requisitos no pertenecen al ámbito de esta investigación. Nuestra intención es que esta RSL se centre en técnicas y procedimientos de transformación, dejando fuera el mercado de herramientas, pues (1) es difícil obtener acceso a todas las herramientas (muchas son propietarias y requieren licencia); (2) hay un vasto número de herramientas CARE; y (3) el mercado de herramientas cambia continuamente. Los informes duplicados del mismo estudio se excluyen también de la RSL: sólo se incluye la versión más completa del estudio. Los trabajos no son excluidos de acuerdo con su fecha de publicación (1) para poder detectar si el tema de la RSL fue investigado activamente en la literatura durante un cierto periodo para luego ser abandonado; y (2) para permitir el descubrimiento de trabajos antiguos que proporcionen ideas que podrían ser adaptadas a técnicas de ingeniería del software actuales.

4.2.1.5 Evaluación de la calidad

En esta RSL la calidad de los estudios seleccionados se evalúa por medio de los siguientes criterios:

- *Lugar de publicación:* a este respecto todas las fuentes seleccionadas buscan sólo revistas y conferencias de prestigio, con la excepción de Google Scholar, que busca en

un espectro de trabajos más amplio. En principio no esperamos seleccionar gran cantidad de trabajos en esta RSL, lo cual nos hace estar abiertos a analizar ideas provenientes de cualquier revista o conferencia, e incluso de literatura gris. Para informar de la calidad del lugar de publicación, optamos por remarcar en el análisis y discusión aquellos estudios que son informes técnicos o *white papers*.

- *Soporte automatizado: s*e estudian las herramientas que soportan cada enfoque, si existen, tratando de evaluar si se trata de prototipos o de herramientas más maduras.
- *Procedimientos de validación:* a cada estudio seleccionado se asigna uno de tres niveles de validación; de menor a mayor: (CEA) el estudio se muestra a través de Casos de Estudio Académicos o incluso a través de ejemplos, que en algunos casos se extraen de la literatura; (CEI) el estudio ha sido puesto en práctica en un Caso de Estudio Industrial; (PI) el estudio puede ser considerado parte de la Práctica Industrial en una organización o en un dominio dados, por ejemplo porque una herramienta comercial que soporta el enfoque está disponible en el mercado. Finalmente, además de esta codificación también se reporta el método de investigación –por ejemplo, Investigación en Acción, en adelante I-A–, si existe.

4.2.1.6 Recogida de datos

Para cada uno de los trabajos seleccionados se rellena una plantilla de extracción de datos adaptada de Biolchini *et al.* [42] (Tabla 44). Esta plantilla consta de una parte de resultados objetivos que se corresponde con lo escrito por los autores del trabajo y de otra parte de resultados subjetivos que se corresponde con las impresiones de los revisores en relación con la temática de esta RSL. La sección de resultados objetivos recopila (1) el método de investigación del estudio (si es reportado); (2) los problemas y limitaciones comentados por los autores (si los hubiera); y (3) un resumen de los resultados del estudio. En este resumen de resultados se presta especial atención a las siguientes cuestiones: (1) las sentencias del trabajo que refuerzan el interés del tema bajo estudio en esta RSL (en relación con la cuestión de la investigación *RQ1*); (2) los modelos iniciales y destino y el método de transformación propuestos en el enfoque (en relación con la cuestión de la investigación *RQ2*); y (3) las herramientas CARE involucradas en el enfoque, junto con los procedimientos de validación reportados (en relación también con la cuestión de la investigación *RQ2*).

Extracción de resultados objetivos	
Identificación del estudio	*Referencia bibliográfica completa*
Origen del estudio	*La fuente a partir de la cual ha sido seleccionado el estudio; la publicación seleccionada a partir de la cual ha sido extraído el estudio, o (prev.RSL) si la publicación pertenece a la línea base de publicaciones previa a la RSL*
Metodología del estudio y validación	*Los métodos de investigación utilizados para conseguir los resultados y los procedimientos de validación, si son reportados*
Resultados del estudio	*Los resultados del estudio relacionados con las cuestiones de la investigación, incluyendo la motivación en la que se basa el trabajo, los modelos iniciales y destino, los procedimientos de transformación y la herramienta CARE, si son reportados*
Problemas y limitaciones del estudio	*Problemas y limitaciones reportados por los autores del estudio*
Extracción de resultados subjetivos	
Impresiones generales y abstracción	*Aquí los revisores incluyen sus propias conclusiones después de la lectura del estudio*

Tabla 44 Formulario de recogida de datos

4.2.1.7 Análisis de datos

Los datos recogidos se han tabulado para mostrar:

- El identificador asignado al estudio en la RSL, sus autores, referencia bibliográfica, fuente y año de publicación.
- La clasificación del estudio siguiendo la taxonomía propuesta (presentada en la Sección 4.2.3).
- El modelo inicial y la clase de sentencias textuales generadas (modelo destino), sea en lenguaje formal o en lenguaje natural (en relación con *RQ2*).
- El método en el cual se ubica el estudio, el soporte automatizado y los eventuales procedimientos de validación (en relación con *RQ2*).

En nuestra opinión, la tabulación de los resultados relacionados con la cuestión de la investigación *RQ1* no es la mejor manera de presentación, pues la motivación es de naturaleza intrínsecamente narrativa. En consecuencia, las justificaciones que consideramos más relevantes acerca del interés de la temática de esta RSL, recogidas de los estudios seleccionados, se presentan en forma narrativa en la Sección 4.2.4.

4.2.2 Resultados de las búsquedas y desviaciones del protocolo

La cadena de búsqueda ha sido adaptada para su uso en cada una de las fuentes. El número de estudios encontrados en cada fuente se resume en la Tabla 45, junto con aquellos marcados como candidatos y aquellos finalmente seleccionados. Como hemos comentado anteriormente, la cadena de búsqueda ha sido formulada necesariamente utilizando palabras de uso común, y por tanto después de aplicar los criterios de inclusión, la mayor parte de los estudios encontrados no han sido etiquetados como candidatos. Los estudios seleccionados se han definido al aplicar los criterios de exclusión a los estudios candidatos. La Tabla 46 muestra los estudios candidatos que no han sido seleccionados y las razones para ello. Un total de 23 estudios fueron así seleccionados (ver Tabla 45). Finalmente, los estudios idénticos encontrados en distintas fuentes han sido eliminados, lo cual ha resultado en un total de 15 estudios seleccionados diferentes.

Después de estudiar la bibliografía de los trabajos seleccionados nos percatamos de una importante corriente relacionada con las cuestiones de la investigación: el *modelado literario* (*literate modeling*, explicado en profundidad en las Secciones 4.2.4 y 4.2.5.1). Ante esta situación pensamos que la RSL no sería completa si esta corriente no se incluyera. Decidimos por tanto introducir una desviación del protocolo, completando el conjunto de estudios seleccionados con las secciones de bibliografía y trabajo relacionado de las 15 contribuciones seleccionadas. De esta manera se seleccionan tres interesantes publicaciones no inicialmente encontradas en la RSL, que se trazan a la bibliografía y trabajo relacionado de los estudios que ya aparecían en la RSL. También han sido seleccionadas seis contribuciones que conocíamos con anterioridad, a pesar de que no aparecían en las búsquedas. Somos conscientes de que se trata de otra desviación del protocolo que amenaza la repetibilidad de la RSL, pero finalmente ha prevalecido el interés de trabajar con un conjunto de trabajos más completo. Así pues, 24 contribuciones han sido finalmente seleccionadas.

Fuente	Estudios encontrados	Estudios candidatos	Estudios seleccionados
IEEE Digital Library	50	2	2
ACM Digital Library	214	8	7
Science@Direct	45	2	2
Meta-Press	87	2	2
Wiley Interscience	5	0	0
Google Scholar	394	12	10
Total	795	26	23

Tabla 45 Número de estudios encontrados, candidatos y seleccionados, por fuente. Los estudios idénticos en distintas fuentes no han sido eliminados todavía

Fuente	Referencia del estudio candidato no seleccionado	Razón para el rechazo
ACM	A. van Lamsweerde. Requirements Engineering in the Year 00: A Research Perspective. in 22nd Intl. Conf. on Softw. Eng. (ICSE'00). 2000. Limerick, Ireland: ACM Press.	[290] (S8 en la Tabla 47) es una versión más reciente del estudio
Google Scholar	B. Jiang, Combining Graphical Scenarios with a Requirements Management System. Master thesis. 2005. University of Ottawa: Ottawa, Ontario, Canada.	Prototipo de herramienta CARE
Google Scholar	N. A. M. Maiden, S. Manning, S. Jones, and J. Greenwood. *Towards Pattern-based Generation of Requirements from Software Model*. in *Requirements Engineering: Foundation for Software Quality 2004 (REFQS'04)*. 2004. Riga, Latvia.	[196] (S3 en la Tabla 47) es una versión más reciente y completa del estudio

Tabla 46 Estudios candidatos no seleccionados

4.2.3 Síntesis de resultados

Las propuestas seleccionadas en esta RSL se disponen en la Tabla 47, que muestra un resumen de los datos de cada estudio seleccionado. Estos datos se analizan de modo general en esta sección. Un análisis más detallado de cada trabajo se encuentra en la Sección 4.2.5.

A modo de síntesis del campo de conocimiento delimitado por las cuestiones *RQ1* y *RQ2*, en esta sección se propone una clasificación de las propuestas seleccionadas en la RSL. La clasificación se basa en el establecimiento de dos dimensiones: *modo de combinación* y *ámbito* (columnas *Modo comb.* y *Ámbito* en la Tabla 47).

En cuanto al modo de combinación, encontramos dos tipos de enfoques que denominamos *generadores* e *integradores*, respectivamente:

- *Modo de combinación: generador:* estos enfoques proponen algoritmos, reglas o patrones para generar requisitos textuales a partir de modelos. Estos trabajos se presentan con o sin herramienta de soporte automático y el texto generado puede ser escrito en lenguaje natural o en algún formalismo.

    - *Generador (lenguaje natural):* estos enfoques generan requisitos candidatos en lenguaje natural, que deben ser validados. En este grupo destacamos el trabajo de Maiden *et al.* sobre generación de requisitos a partir de modelos i* [196] (S3 en la Tabla 47); la investigación de Meziane *et al.* [215] (S13), para generar especificaciones en inglés que parafrasean diagramas de clases UML; la propuesta de Firesmith [105] (S19) que deriva requisitos textuales a partir de casos de uso, escenarios e historias de usuario; y la propuesta de Berenbach [38] (S17) que estrictamente no general texto en lenguaje natural sino más bien una jerarquía de requisitos a partir de diagramas de casos de uso.

  - *Generador (lenguaje formal):* estos enfoques generan requisitos escritos en una notación formal. Los métodos formales proporcionan beneficios como el análisis mecánico de un sistema para comprobar la ausencia de interbloqueos (como *deadlock* y *livelock*), pero la adopción de métodos formales en la industria se encuentra comprometida por el coste y la complejidad de la especificación formal del sistema. De ahí que algunas propuestas de la literatura investiguen la derivación de especificaciones formales a partir de modelos de requisitos. En este grupo destacamos el trabajo en relación con modelos de metas realizado por van Lamsweerde *et al.*, que se dirige a la generación automática de requisitos operacionales descritos por medio de pre y postcondiciones y disparadores [183] (S5), y la generación de requisitos descrita por medio de la técnica tabular SCR [83] (S7). Además, Cabral y Sampaio [50] (S20) investigan la generación de requisitos operacionales en un álgebra de procesos en CSP a partir de especificaciones de casos de uso.

- *Modo de combinación: integrador:* estos estudios no proporcionan algoritmos, reglas o patrones para generar requisitos a partir de modelos, sino más bien una clase de *guías* para relacionar modelos y requisitos textuales (o, en ocasiones, más bien una *filosofía de trabajo*). Los requisitos resultantes pueden estar escritos en lenguaje natural o en una notación formal. Esta RSL se concentra particularmente en los enfoques generadores, si bien los enfoques integradores pueden ser de interés en la generación de requisitos y de documentos de requisitos y han sido por tanto también incluidos. Algunos ejemplos de esta categoría vienen dados por la propuesta de Arlow [30] (S1) sobre modelado literario y la visión de Firesmith de cómo debe ser una moderna especificación de requisitos [104] (S2).

En cuanto al *ámbito* del enfoque, nos encontramos con propuestas que se centran exclusivamente en la generación de requisitos individuales (correspondencia *m2rs*, descrita en la Sección 4.2.1.1), mientras que otras propuestas abordan la creación de documentos de requisitos (correspondencia *m2rd*, Sección 4.2.1.1). Obviamente, esta dimensión no es disjunta, dado que un mismo estudio puede tratar la generación de requisitos individuales y la generación de documentos de requisitos. En relación con esta RSL, ambos enfoques son de igual interés:

- *Ámbito: requisito:* estas propuestas versan sobre la generación de requisitos o de conjuntos de requisitos, pero no necesariamente sobre los documentos de requisitos en los cuales se pueden ubicar los requisitos. Un ejemplo es el enfoque de Maiden *et al.* [196] (S3), que concierne la derivación de requisitos desde modelos SD de i*.
- *Ámbito: documental:* estos estudios se centran en la generación manual, automática o semiautomática de documentos de requisitos. Por ejemplo, el grupo de van Lamsweerde [290] (S8) ha desarrollado una herramienta (denominada *Objectiver*) con la cual se puede generar semiautomáticamente documentos de requisitos estructurados a partir de una especificación de metas. La corriente de modelado literario como un todo también se puede incluir en el ámbito documental, como por ejemplo los trabajos de Arlow [30] y de Firesmith [104] (estudios S1 y S2 en la Tabla 47).

A partir de los resultados de la Tabla 47, la Figura 35 relaciona el modo de combinación con el ámbito, distinguiendo entre resultados en lenguaje natural y en lenguaje formal. Observando esta figura se puede concluir que se ha prestado más atención a la generación de requisitos individuales (24 estudios) que a la de documentos (diez), mientras que las propuestas

generadoras (23 estudios) son más numerosas que las integradoras (11). En relación con los requisitos generados, es decir, considerando la columna *Texto generado* en la Tabla 47, se observa como la mayor parte de las estrategias estudiadas tratan con lenguaje natural más que con lenguaje formal (26 estudios a ocho). Esta columna muestra también si los estudios correspondientes reportan el uso de alguna plantilla para la especificación de requisitos en lenguaje natural (como por ejemplo Volere) o de alguna notación formal (como por ejemplo SCR, KAOS o Albert). Nótese que el mismo estudio puede estar dirigido a requisitos y a documentos, y a lenguaje natural y formal.

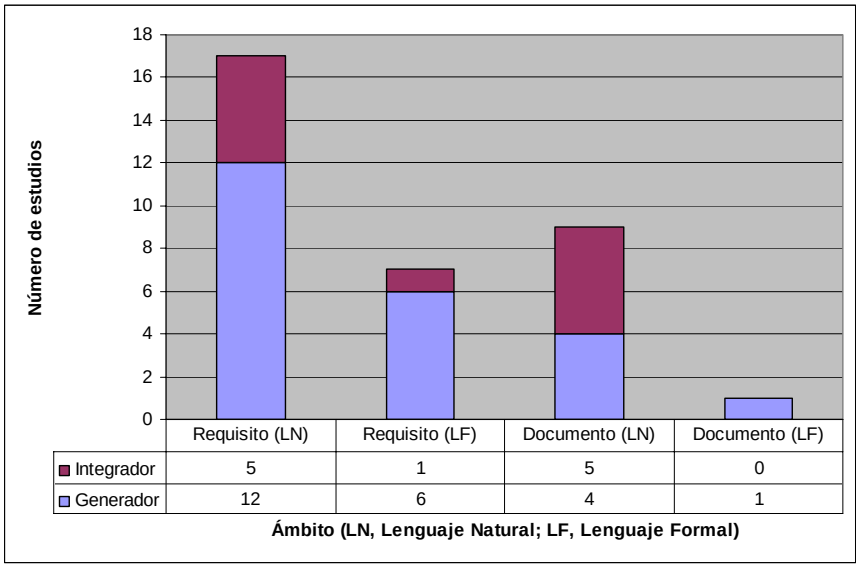

Figura 35 Número de estudios por ámbito y modo de combinación

Los estudios recogidos en la Tabla 47 se aplican a un número considerable de modelos (columna *Modelo inicial*), que se resumen en la Figura 36. Esta figura no computa los enfoques de modelado literario (S1 y S2 en la Tabla 47) porque son aplicables a cualquier lenguaje de modelado visual. Es particularmente destacable el número de estudios relacionados con casos de uso y escenarios (diez estudios, siete de ellos generadores) y los modelos de metas (siete estudios, cinco de ellos generadores). Por un lado, aunque casos de uso y escenarios disponen de notaciones gráficas útiles y bien conocidas, son técnicas en las cuales el texto juega tradicionalmente un rol más importante que los diagramas. Por otro lado, los modelos de metas se benefician de la atención que van Lamsweerde *et al.* prestan a KAOS (cuatro estudios generadores, S4-5 y S7-8 en la Tabla 47). Dos enfoques se relacionan con el modelado del negocio, pero ninguno de ellos considera la generación de documentos. Hay también dos estudios sobre modelos de interfaz de usuario. Merece la pena destacar que sólo se ha encontrado una propuesta generadora sobre UML, dejando a un lado casos de uso y escenarios. Parece que se debería realizar más esfuerzo en esta área, particularmente en aquellas técnicas de UML que son más adecuadas para la IR.

Con respecto a la columna *Método* en la Tabla 47, pocos estudios forman parte de un método de desarrollo de software concreto, lo cual no es una sorpresa pues pensamos que cada propuesta puede ser utilizada en el contexto de cualquier método en el cual se utilicen sus modelos iniciales correspondientes.

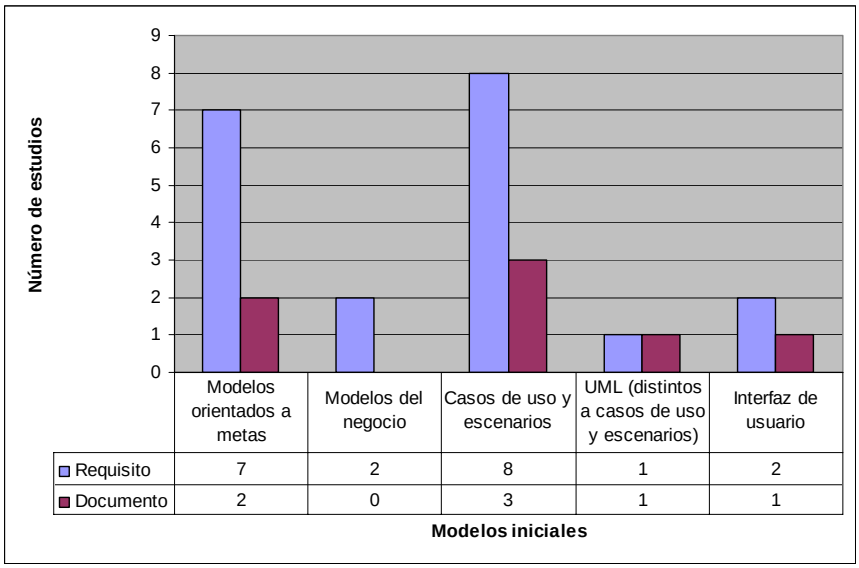

Figura 36 Número de estudios por ámbito y modelo de partida

El primer criterio de evaluación de la calidad establecido en la Sección 4.2.1.5 es *Lugar de publicación*. Entre los estudios seleccionados solamente aparece un *white paper*, uno de Probasco y Leffingwell [253] (S22). Parece que el resto de estudios ha sido publicado después de un proceso de revisión, con la posible excepción de dos columnas de Firesmith en el JOT [104, 105] (S2 y S19). El capítulo de libro de Arlow y Neustadt sobre modelado literario [30] (S1) procede de una contribución a una conferencia [29]. En la Tabla 47 se recogen también los datos en relación con *Herramienta* y *Validación*:

- Considerando *Herramienta*, en la medida de nuestro conocimiento, REDEPEND y Objectiver son las herramientas más maduras que se han encontrado en esta RSL. Objectiver está incluso disponible comercialmente. Además, la automatización no es reportada en ciertos estudios, de forma destacada en la propuesta de Arlow y Neustadt [30] (S1) sobre modelado literario y en la *visión* de Firesmith sobre una moderna especificación de requisitos [104] (S2). El resto de enfoques que reportan automatización muestran su viabilidad por medio de prototipos cuyo nivel de madurez real es difícil de valorar sobre la base de la información aportada por las publicaciones.
- La Figura 37 presenta un resumen de *Validación*, y pone de manifiesto que la práctica industrial real es escasa en este campo. Un caso de estudio en una organización debe superar un salto importante si se trata de convertir en parte de la práctica real de la organización. La lectura de las contribuciones no permite en todos los casos valorar con precisión el tamaño de este salto. El número de casos de estudio académicos e industriales es similar (nueve y 11 estudios, respectivamente). En principio los casos de estudio industriales son preferibles, aunque con frecuencia es difícil realizar una valoración precisa del ámbito real de un caso de estudio industrial a través de la mera lectura de las publicaciones. No obstante, algunos trabajos presentan una investigación teórica rigurosa y sin embargo están ilustrados por medio de un caso de estudio académico (por ejemplo los estudios S4, S5 y S7 en la Tabla 47, realizados por van Lamsweerde *et al.*).

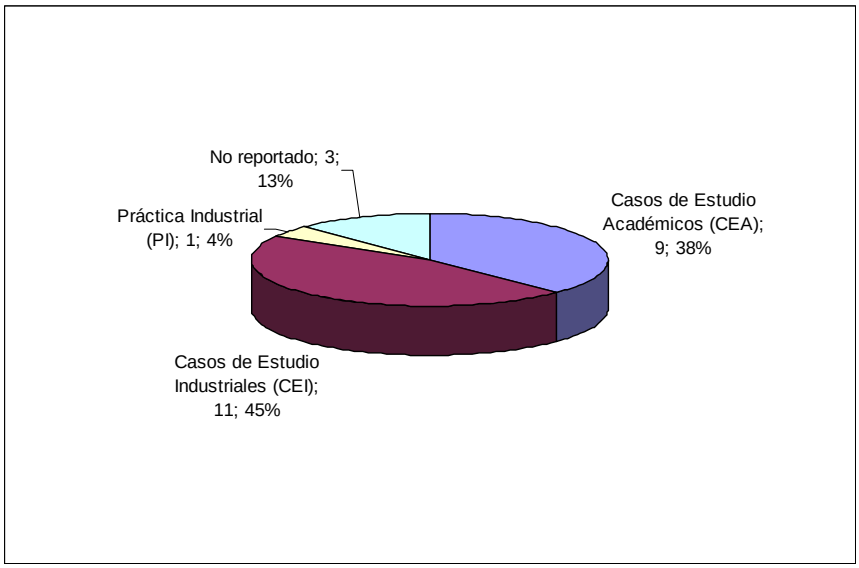

Figura 37 Tipos de validación (se incluye número de estudios y porcentaje)

Finalmente, aunque no por ello de menor importancia, cabe decir que no existe ninguna propuesta que considere la sincronización entre los documentos o requisitos individuales generados y los modelos de partida. En los estudios seleccionados la generación siempre se lleva a cabo en una dirección, de modelos a requisitos. No existe ningún enfoque que posibilite que los cambios en los requisitos generados se propaguen automáticamente a los modelos iniciales. Pensamos que esta sincronización podría ser útil especialmente en un proceso software iterativo e incremental, en particular durante la validación con los clientes. Por tanto la validación se podría llevar a cabo directamente en los requisitos textuales generados, ampliamente comprensibles por todos los interesados cuando están formulados en lenguaje natural. Estos requisitos podrían ser cambiados y los modelos relacionados evolucionarían automáticamente a través de relaciones de traza. Con esto nos referimos, claro está, a la sincronización de ciertos cambios predefinidos en los requisitos textuales o en los documentos generados, puesto que la sincronización de cambios en general puede ser muy difícil.

Después de haber rellenado los formularios de recolección de datos y de haber analizado las contribuciones seleccionadas, a continuación se responde las cuestiones de la investigación presentadas en la Sección 4.2.1.2.

4.2.4 Discusión RQ1: ¿Qué valor se encuentra en la literatura en cuando a la generación de especificaciones de requisitos desde modelos de ingeniería del software?

En las secciones 1.1.4 y 4.1 se formula una justificación a priori de este tópico, antes de que la RSL se realizara. Esta cuestión (*RQ1*) nos permite buscar afirmaciones en la literatura que justifiquen el interés en la temática de la RSL. En esta sección se recogen las afirmaciones que consideramos más representativas. Arlow y Neustadt [30] (S1 en la Tabla 47) opinan que en muchos casos no todos los interesados son capaces de comprender la sintaxis y la semántica de los modelos de negocio expresados en UML u otras técnicas visuales. Más aún, estos autores discuten las siguientes cuestiones en relación con los modelos visuales:

ID	Autor/es [ref.] (fuente)	Año	Modo comb.		Ámbito		Modelo inicial	Texto generado		Método	Herramienta	Validación
			Gen.	Int.	Req.	Doc.		LF	LN			
Sección 4.2.5.1. Modelado Literario												
S1	Arlow y Neustadt [30] (muy similar y más reciente que [29], que es citado por S12)	2004	□	■	■	■	UML o cualquier otro lenguaje visual	□	■	No específico	No reportado	(CEI) Modelos de objetos del negocio en British Airways
S2	Firesmith [104] (prev. RSL)	2003	■	□	■	■	No específico	□	■	No específico	No reportado	No reportado *(vision paper)*
Secciones 4.2.5.2 – 4.2.5.4. Enfoques de IR orientada a objetivos												
S3	Maiden *et al*. [196] (acm, mpress, gs)	2005	■	□	■	□	Modelo orientado a objetivos i* (diagrama SD)	□	■ Volere	RESCUE	REDEPEND	(CEI) (I-A) DMAN, gestión del tráfico aéreo
S4	van Lamsweerde y Willemet [291] (acm, gs)	1998	■	□	■	□	Escenarios y casos de uso	■ KAOS	□	KAOS	Sí (todavía no incluida en GRAIL)	(CEA) Cajero automatico y ascensores
S5	Letier y van Lamsweede [183] (citado por S6 y S7)	2002	■	□	■	□	Modelo orientado a objetivos KAOS	■ Modelo operacio-nal	□	KAOS	Validación con SteP verif. system	(CEA) Sistema de control de bombeo
S6	Alrajeh *et al.* [24] (acm)	2006	■	□	■	□	Modelo orientado a objetivos con lógica temporal	■ Modelo operacio-nal	□	No específico	Validación con *model checker* LTSA y herram. de aprendizaje inductivo Progol5	(CEA) Precondiciones en los modelos KAOS de un sistema de control de bombeo
S7	Landtsheer *et al.* [83] (acm, mpress, gs)	2004	■	□	■	□	Modelo orientado a objetivos KAOS	■ SCR	□	KAOS	Validación con *model checker* SMV	(CEA) Sistema de inyección segura para una planta de energía nuclear
S8	van Lamsweerde [290] (citado por S3)	2004	■	□	■	■	Modelo orientado a objetivos KAOS	■ Modelo operacio-nal	■	KAOS	CASE comercial Objectiver (antes prototipo GRAIL)	(PI) Alrededor de 20 proyectos industriales; Objectiver es herramienta comercial
S9	Yu *et al.* [300] (gs)	1995	□	■	■	□	Modelo orientado a objetivos i*	■ Albert	□	No reportado	No reportado	(CEA) Sistema bancario
S10	Antón y Potts [27] (prev. SLR)	1998	□	■	■	■	Modelo orientado a objetivos	□	■	GBRAM	No reportado	(CEI) (I-A) Web CommerceNet
Sección 4.2.5.5. Modelado del negocio												
S11	Cox *et al.* [61] (sd, gs)	2005	□	■	■	□	Modelo del negocio (RAD)	□	■	No específico	No reportado	(CEI) (I-A) Sistema comercio electrónico
S12	Türetken *et al.* [287] (gs)	2004	■	□	■	□	Modelo del negocio (eEPC)	□	■	No específico	"KAOS" plugin para toolkit ARIS	(CEI) Dos aplicaciones militares
Sección 4.2.5.6. Enfoques basados en UML												
S13	Meziane *et al.* [215] (prev. SLR)	2007	■	□	■	■	Diagrama de clases UML	□	■	Cualquier método basado en UML	Prototipo Java GeNLangUML	(CEA) Sistema universitario

ID	Autor/es [ref.] (fuente)	Año	Modo comb.		Ámbito		Modelo inicial	Texto generado		Método	Herramienta	Validación
			Gen.	Int.	Req.	Doc.		LF	LN			
Sección 4.2.5.7. Modelado de casos de uso, escenarios e historias de usuarios												
S14	Maiden *et al.* [192] (ieee)	1998	■	□	■	□	Escenarios y casos de uso	□	■	CREWS-SAVRE	CREWS-SAVRE	(CEI) Servicio de ambulancias de Londres
S15	Mavin y Maiden [212] (ieee, gs)	2003	■	□	■	□	Escenarios y casos de uso	□	■	CREWS-SAVRE	CREWS-SAVRE	(CEI) OCD (naval) & CORA-2 (gestión de tráfico aéreo)
S16	Maiden y Robertson [194] (acm)	2005	■	□	■	□	Escenarios y casos de uso	□	■ Volere	RESCUE (evoluciona CREWS-SAVRE)	ART-SCENE (evoluciona CREWS-SAVRE)	(CEI) (I-A) DMAN, gestión de tráfico aéreo
S17	Berenbach [38] (acm, gs)	2003	■	□	■	□	Diagramas de casos de uso	□	■	Cualquier proceso dirigido por casos de uso	Basado en *scripts* de herramienta CASE	(CEI) Modelos de casos de uso usados en Siemens
S18	Berenbach [39] (prev. SLR)	2004	□	■	■	■	Diagramas de casos de uso	□	■	Cualquier proceso dirigido por casos de uso	No reportado	(CEI) Sistema de ordenación de correo en Siemens
S19	Firesmith [105] (gs)	2004	■	□	■	□	Historias, escenarios y casos de uso	□	■	Cualquier proceso dirigido por casos de uso	No automatizado	(CEA) Ejemplo cajero automático
S20	Cabral y Sampaio [50] (sd, acm)	2008	■	□	■	□	Plantillas de casos de uso	■ CSP	□	No específico	Ms Word plugin, traductor CNL/CSP, FDR, CSP *model checker*	(CEI) Cooperación de investigación con Motorola
S21	Daniels *et al.* [81] (prev. SLR)	2005	□	■	■	■	Casos de uso	□	■	Rational Unified Process	No automatizado	(CEA) Ejemplo línea de productos microondas
S22	Probasco y Leffingwell [253] (gs)	1999	□	■	□	■	Casos de uso	□	■	Rational Unified Process	Rational Suite	No reportado (*white paper*)
Sección 4.2.5.8. Modelado de la interfaz de usuario												
S23	Jungmayr y Stumpe [157] (gs)	1998	■	□	■	■	Modelos de uso extendido	□	■	No específico	Prototipo Java	(CEA) Consulta de bases de datos bibliográficas UNIrech
S24	Smith [273] (prev. SLR)	1982	■	□	■	□	Modelo de interfaz usuario-sistema	□	■	No específico	Prototipo sobre UNIX	No reportado

Leyenda: *fuente*: acm, ieee, sd (ScienceDirect), mpress (MetaPress), wi (Wiley Interscience), gs (Google Scholar), prev.RSL (línea base previa a la RSL), citado por [ref.];
Modo comb. (Modo de combinación): Gen: Generador / Int: Integrador; *Ámbito*: Req: Requisito / Doc: Documento
Texto generado: LF: Lenguaje Formal / LN: Lenguaje Natural;
Validación: (CEA) Caso de Estudio Académico / (CEI) Caso de Estudio Industrial / (PI) Práctica Industrial; I-A: Investigación en Acción

Tabla 47 Estudios de la revisión sistemática considerando RQ2

- En orden a acceder a la información embebida en un modelo puede ser necesario conocer cómo operar una herramienta de modelado. Los informes que toda herramienta de modelado genera –normalmente en formato HTML– son con frecuencia difíciles de leer y de navegar y tienen por tanto, según la experiencia de Arlow y Neustadt, un uso práctico limitado.
- A menos que uno sea familiar con la "forma" general de un modelo visual, puede ser difícil determinar dónde comenzar la lectura, cuando se lee el modelo en una herramienta o en un informe.
- Es a veces difícil, o incluso imposible, descubrir los requisitos del negocio y la motivación que subyace en el modelo visual, pues los requisitos del negocio se convierten en *invisibles* cuando se llevan fuera del contexto del negocio y se expresan en una notación visual. Por ejemplo, un requisito altamente importante puede ser expresado de manera tan concisa que puede ser fácilmente pasado por alto durante un recorrido del modelo (*walkthrough*). Arlow *et al.* denominan a esto "la trivialización de los requisitos del negocio por el modelado visual" [30]. Estos autores muestran esta *trivialización* a través de un ejemplo en el cual un importante requisito del negocio se expresa finalmente en los modelos como una multiplicidad "n" en una asociación en lugar de "1": este requisito podría fácilmente pasar desapercibido durante la validación del modelo.

El modelado literario se inspira en la programación literaria (*literate programming*) propuesta por Knuth en los años ochenta. El modelado literario es una propuesta en la cual el modelado y la documentación en lenguaje natural están ligados entre sí sin costuras y las funcionalidades de las herramientas CASE (*Computer-Aided Software Engineering*) y de las herramientas CARE están verdaderamente integradas. Creemos que muchos ingenieros de software han usado de alguna manera el modelado literario a lo largo del ejercicio de su profesión, de manera *ad hoc*, sin ser conocedores de este término. En un trabajo referenciado por Türetken *et al.* [287] (S12), Finkelstein y Emmerich [103] examinan el futuro de las herramientas CARE y colocan el modelado literario en el futuro a largo plazo de estas herramientas. Estos dos autores afirman que no hay todavía una manera adecuada de usar de forma sinérgica los modelos y el lenguaje natural en las herramientas CARE, y ven el *modelado literario* en el futuro de la tecnología CARE. Existe una corriente que considera que el lenguaje natural es un vestigio del pasado que se mantiene debido a la falta de transferencia tecnológica de la investigación a la industria en cuanto a métodos de modelado. Sin embargo, Finkelstein y Emmerich piensan que el lenguaje natural juega un papel valioso en la especificación de requisitos y que probablemente no va a ser suplantado. El lenguaje natural sirve para ligar los componentes de los modelos con los fenómenos del mundo real, y permite a los interesados validar las especificaciones más fácilmente. Meziane *et al.* [215] (S13) añaden otro beneficio: la generación automática de requisitos en lenguaje natural con el objetivo de mejorar el mantenimiento. Con frecuencia la implementación del software no es consistente con la documentación, pues los desarrolladores no actualizan los modelos de análisis y diseño cuando cambian el código. Los modelos de diseño pueden ser generados a partir de una implementación que evoluciona por medio de una herramienta de ingeniería inversa. Para Meziane *et al.* sería útil si se generaran nuevos requisitos en lenguaje natural de acuerdo con el diseño actualizado.

Maiden *et al.* [196] (S3) manifiestan que su trabajo se enmarca dentro de "una importante corriente hacia la integración de modelos de requisitos". Estos autores

afirman que “aunque existen muchas técnicas de especificación y análisis de requisitos producidas por la investigación académica, la mayor parte de las organizaciones continúan documentado los requisitos de forma textual” y que “desafortunadamente, la mayor parte de las aproximaciones de modelado no han sido diseñadas para soportar la derivación de requisitos textuales desde los modelos, o para ser usadas junto con tales descripciones textuales de requisitos.”

Cuando van Lamsweerde [290] (S8) expone las lecciones aprendidas en la aplicación de la IR basada en metas, comenta que “la diversidad de proyectos de IR en cuanto a tipo, tamaño y enfoque demanda tecnologías altamente flexibles. Pensamos que un método y herramienta (…) que por defecto soporte especificaciones gráficas y textuales, además de especificaciones formales sólo cuando y donde se necesitan para el análisis incremental de fragmentos críticos de modelos, es un paso prometedor en la buena dirección”. van Lamsweerde afirma también que muchas veces los documentos de requisitos se perciben como grandes, complejos, no actualizados y demasiado lejanos de los productos ejecutables por los cuales los clientes están dispuestos a pagar. Este autor añade que “al final, lo que a los clientes importa más es la calidad de las entregas del proyecto. El documento de requisitos dirigido por modelos generado con nuestra herramienta [Objetiver] ha sido percibido como el principal indicador de éxito en muchos proyectos.”

Firesmith [104] (S2) discute los problemas asociados con la especificación de requisitos tradicional y propone un conjunto de requisitos que debería satisfacer un enfoque para gestionar tales problemas en un proceso software iterativo e incremental. Tal aproximación debería permitir la generación automática de especificaciones de requisitos adaptadas a sus usuarios. Por ejemplo, son muy distintas las necesidades de los directores ejecutivos o del director del proyecto de las de los probadores del software. En esta línea relacionada con modelos de proceso iterativos, en la vía formulada por Antón y Potts [27] (S10) sobre IR basada en metas se resalta la importancia de los documentos de requisitos con anotaciones de cuestiones no resueltas. En un documento de requisitos puede ser tan importante listar requisitos como registrar cuestiones a estudiar, que van apareciendo y desapareciendo dinámicamente.

Türetken *et al.* [287] (S12) trabajan en la generación automática de requisitos en lenguaje natural a partir de modelos de procesos del negocio y afirman que mediante su aproximación se consigue minimizar el tiempo en “tareas sin valor añadido” como reescribir y documentar requisitos, mejorando así la calidad de los documentos de requisitos y facilitando su modificación. Estos autores afirman que los requisitos funcionales del sistema y/o del software están generalmente basados en modelos visuales, pero normalmente la trazabilidad y la integración entre estos modelos y los requisitos especificados en lenguaje natural no están establecidas y se pierden. Uno de los beneficios de este enfoque es que el formato estándar de los requisitos generados los hace más adecuados para la comprensión, validación, verificación y gestión por parte de todos los interesados.

Como resultado de su experiencia en Siemens, Berenbach’s [38] (S17) ha observado “la desconexión entre UML y los requisitos de los procesos modelados”. Este autor comenta que el salto se acentúa conforme los modelos se hacen más complejos: la extracción de requisitos detallados se hace más difícil. Y además si no se desarrolla un conjunto completo de requisitos a partir de los modelos UML en el desarrollo posterior

puede haber problemas. Por ejemplo, los casos de prueba derivados pueden no proporcionar una cobertura completa. Berenbach [39] (S18) afirma que la solución viene dada por la integración planificada de los modelos, el texto de los casos de uso y los requisitos, y proporciona indicaciones de cómo se debe producir esta integración. Para Berenbach un modelo UML es un repositorio, no un conjunto de diagramas. Este autor también proporciona una lista de las "mejores prácticas", entra las que cita la "generación de documentación bajo demanda".

En cuanto a la integración de casos de uso y requisitos textuales, Daniels *et al.* [81] (S21) establecen que los casos de uso aportan facilidad de comprensión, contexto, y trazabilidad directa hacia las necesidades de un actor, mientras que los requisitos textuales del tipo *El sistema deberá…* (*shall-statement requirements*) añaden la precisión necesaria para especificar el sistema de forma completa y sin ambigüedades. A pesar de la utilidad de los casos de uso, otras formas de especificación como requisitos textuales, diagramas, tablas, ecuaciones, grafos, imágenes, pseudocódigo y máquinas de estado deben ser utilizadas para capturar requisitos adicionales y para proporcionar el nivel de detalle apropiado para caracterizar un sistema. Estos autores concluyen que "los modelos de casos de uso y los requisitos textuales tradicionales tipo *El sistema deberá…* son técnicas de especificación sinérgicas que deberían ser utilizadas de forma complementaria para comunicar y documentar mejor los requisitos."

4.2.5 Discusión RQ2: ¿Qué técnicas se han considerado en este campo?

Por motivos de claridad, las propuestas seleccionadas, que se describen en esta sección, siguen el mismo modo de agrupación que en la Tabla 47.

4.2.5.1 Modelado literario

En la Sección 4.2.4 se incluye una introducción detallada de los objetivos generales del modelado literario. La propuesta original de Arlow *et al.* [29] sobre este tópico se refina posteriormente en [30] (S1 en la Tabla 47). En este enfoque se define un nuevo tipo de documento, *Documento de contexto del negocio* (*Business Context Document, BCD*), en el cual (1) los diagramas quedan incluidos como figuras en un documento escrito en lenguaje natural, en el que se parafrasea el modelo visual (por ejemplo, en relación con una determinada asociación UML se podría decir "Cada Producto tiene una ListaDePrecios que contiene cero o más Precios"); (2) se incluyen breves descripciones de la sintaxis de UML como notas al pie; (3) se ponen ejemplos de la notación, ejemplos sencillos pero reales para ilustrar aspectos específicos; y (4) se destacan cuestiones relevantes como la motivación (*rationale*). Los modelos literarios son así modelos UML embebidos en texto en lenguaje natural que los explica.

Según la experiencia de Arlow *et al.*, es preferible estructurar el BCD en torno a las *cosas* que proporcionan valor al negocio más que en torno a los *procesos*. Las cosas son más estables que los procesos y tienden a formar de forma natural clusters cohesivos que proporcionan un enfoque adecuado para el BCD. Se debería poder establecer además una relación sencilla entre la estructura de los paquetes del modelo UML y el BCD. La propuesta de Arlow *et al.*, sin embargo, no menciona patrones, reglas o algoritmos específicos para generar requisitos a partir de modelos UML: en lugar de ello, estos autores describen los contenidos básicos del BCD y proporcionan guías para

crearlo. De acuerdo con Arlow *et al.*, cada BCD debería tener la siguiente estructura: (1) Contexto del negocio; (2) Cumplimiento de estándares; (3) Un *modelo guía de UML*, que mostrara las principales *cosas* y sus relaciones, con relaciones cruzadas hacia las partes apropiadas del BCD; y (4) un número de secciones, cada una con la descripción de una cosa o de varias cosas relacionadas, y englobando: (4.1) una parte de prosa o narrativa, que referencia uno o varios fragmentos de modelos; (4.2) un conjunto de diagramas UML que ilustran la prosa o narrativa anterior (de acuerdo con estos autores, se trata típicamente de diagramas de clases, diagramas de casos de uso y diagramas de secuencia); y (4.3) diagramas informales, allí donde mejoren la descripción del negocio. Arlow *et al.* proporcionan también guías sobre el estilo de escritura que debe ser usado en el BCD y, por ejemplo, recomiendan el uso de ejemplos concretos para ilustrar los modelos y el desarrollo de un vocabulario del negocio.

Creemos que, en sentido amplio, las propuestas seleccionadas y analizadas en esta RSL pueden ser consideradas parte de esta corriente de modelado literario, aunque en las publicaciones no se haga referencia a esta vía. En particular, nos parece que la propuesta de Firesmith en relación con las especificaciones de requisitos [104] (S2) puede ser considerada un buen ejemplo de modelado literario. El objetivo de este enfoque es la obtención de una especificación de requisitos correcta, completa, consistente, actualizada y apropiada a su destinatario (es decir, que ayude a soportar las tareas de cada uno de sus posibles lectores). Para mejorar la especificación de requisitos, a partir de su experiencia Firesmith propone el uso de una herramienta CARE que (1) tenga un repositorio de requisitos de grano fino; (2) permita la generación automática de especificaciones, como una especie de separación entre el modelo y la vista en el paradigma MVC (*Modelo-Vista-Controlador*); y (3) genere diferentes especificaciones para diferentes lectores. Otra idea interesante que Firesmith apunta es la necesidad de establecer algún tipo de mecanismo publicar-suscribir (*publish-subscribe*) que permita que los interesados sean notificados cuando ciertos requisitos de interés para ellos cambien en una iteración o sean añadidos en un desarrollo incremental. Esta característica sería clave, por ejemplo, para un diseñador.

En conclusión, en palabras de Arlow y Neustadt [30]: “en la práctica, el modelado literario, aunque es una idea muy buena, no ha llegado a despegar adecuadamente. Esto ha sido debido en parte a que confía en herramientas especiales de proceso de textos, que no son ampliamente disponibles, y en parte porque los programadores generalmente prefieren escribir código a prosa. En contraste con esto, el modelado literario ha probado ser muy útil para aquellos que lo han intentado. Esto es debido a que un modelo literario no solamente proporciona un contexto para un modelo UML que de otra manera carecería de dicho contexto, sino porque también ayuda al modelador a hacer su trabajo”. Nosotros pensamos que para que la propuesta de modelado literario sea un éxito es preciso que esté basada en la generación automática o fuertemente asistida del BCD o de partes significativas de él. Si el BCD se debe crear manualmente en su totalidad, entonces probablemente el modelado literario no se aplicará en la práctica: la generación manual del BCD puede ser engorrosa, y los BCDs pueden ser difíciles de escribir porque, de acuerdo con Arlow y Neustadt, requieren una visión del negocio amplia y consistente, conocimiento adecuado de UML y buenas destrezas de comunicación oral y escrita.

4.2.5.2 RESCUE y REDEPEND: generación de requisitos candidatos a partir de i*

En el conjunto de propuestas de Neil Maiden *et al.* en relación con el proceso de IR RESCUE, una de las premisas es que para los interesados es más sencillo detectar un error de comisión que uno de omisión. En otras palabras, para un interesado es más sencillo reconocer un posible escenario alternativo en un caso de uso o un requisito candidato en lenguaje natural y aceptarlo o rechazarlo que evocar todos los escenarios posibles o todos los requisitos del sistema. En este contexto, en el ámbito de RESCUE se pueden encontrar dos conjuntos de trabajos: (1) aquellos que generan requisitos en lenguaje natural a partir de modelos de contexto SD (*Strategic Dependency*) de i* (estos modelos SD son básicamente diagramas de contexto que permiten capturar las metas de la interacción entre los agentes de un sistema sociotécnico); y (2) aquellos que tratan de completar el conjunto de escenarios alternativos de los casos de uso (descritos en la Sección 4.2.5.7).

En el primer conjunto de publicaciones, Maiden *et al.* [196] (S3) describen 19 patrones desarrollados para *parafrasear* un modelo SD de i*, haciendo así explícita información que está especificada en el modelo. Estos patrones identifican estructuras sintácticas y semánticas que aparecen recurrentemente en los modelos i*, a partir de las cuales se generan requisitos candidatos que deben ser validados. Estos requisitos se especifican en lenguaje natural mediante la plantilla VOLERE. Este enfoque ha sido aplicado manualmente en un caso de estudio industrial, DMAN (*DEparture MANager*), un sistema de gestión del tráfico aéreo. Los autores consideran un éxito esta propuesta ya que en tres días se generaron 214 requisitos textuales, de los cuales 207 fueron incluidos en la especificación de requisitos final (que engloba casi 900 requisitos). Por tanto, casi un 25% de los requisitos de la especificación final se generaron de esta forma. Con esta aproximación se pretende mejorar la completitud de la especificación de requisitos aunque, como apuntan Regnell *et al.* [258], probablemente no de la extracción en sí misma, ya que la información de partida debe estar contenida en el modelo i*. Este enfoque puede ser útil en cualquier sistema en el cual un conjunto heterogéneo de actores interactúa para conseguir ciertas metas. Maiden *et al.* concluyen este estudio preguntándose si una generación automática de sentencias de requisitos candidatos, basada en estos patrones, sería efectiva en coste, o si el número de requisitos duplicados y de falsos positivos sería inaceptable.

Una respuesta inicial a la última cuestión planteada por el estudio S3 en el párrafo anterior se puede encontrar en una publicación posterior, [191], en la cual Maiden *et al.* presentan las lecciones aprendidas durante la aplicación de i* a varios casos de estudio industriales, y discuten el uso de una nueva versión de su herramienta REDEPEND, que automatiza la aplicación de los patrones anteriores. En relación con el propósito de esta RSL, durante un taller de requisitos REDEPEND permite a los analistas construir un modelo SD de i*, generar automáticamente en tiempo real requisitos candidatos a partir de este modelo de i*, y recorrer estos requisitos candidatos para seleccionarlos o rechazarlos. Los 19 patrones anteriormente mencionados se representan en REDEPEND haciendo uso de un archivo Ms Excel, con lo cual se puede añadir cualquier eventual nuevo patrón sin necesidad de cambiar la herramienta. En el caso de estudio de DMAN, citado anteriormente, la generación automática de 287 requisitos candidatos necesitó solamente de 12 segundos en un PC estándar. El número de requisitos candidatos generados es mayor que en [196] debido a que los 19 patrones originales habían sido

refinados, obteniéndose así más requisitos de distinto tipo. En relación con la generación de requisitos, la evaluación inicial que Maiden *et al.* realizan de la herramienta es positiva, aunque pretenden recopilar más datos de evaluación.

4.2.5.3 Ingeniería de requisitos orientada a objetivos con KAOS y Objectiver

Axel van Lamsweerde *et al.* han producido un sólido conjunto de trabajos que se articulan alrededor de KAOS, un marco orientado a metas que incluye un método y soporte automatizado. El método KAOS ha sido aplicado en un número considerable de dominios distintos en alrededor de 20 proyectos industriales en el CEDITI (un *spin-off* universitario) [290]. En KAOS se usa lenguaje natural para describir informalmente el sistema, aunque se puede hacer uso de una lógica temporal, allí donde sea preciso, para describir el sistema formalmente. Esta sección se ocupa de la generación tanto de requisitos textuales individuales como de documentos de requisitos. En relación con los requisitos individuales, los requisitos generados no se especifican mediante lenguaje natural, sino a través de notaciones formales (pre y postcondiciones y disparadores, por un lado, y SCR, por otro). Los modelos de partida consisten en escenarios y modelos de metas, que disponen de una útil representación diagramática, aunque en esta propuesta son procesados a partir de su representación textual.

El primer trabajo en esta sección está relacionado con escenarios, que son ampliamente considerados un medio efectivo para extraer, validar y documentar requisitos. Sin embargo, los escenarios son parciales y pueden pasar por alto o dejar implícitas ciertas propiedades del sistema. Además, las metas, requisitos y asunciones relacionadas con los escenarios se describen sólo implícitamente. Por tanto, las propiedades del sistema deben ser expresadas explícitamente de manera que se pueda llevar a cabo un análisis de consistencia y completitud. Debido a todo esto, en el ámbito de KAOS se ha desarrollado un método para inferir de forma sistemática especificaciones formales, abstractas y declarativas de metas, requisitos y asunciones a partir de escenarios informales en lenguaje natural [291] (S4). La especificación generada utiliza lógica temporal. Esta propuesta S4 está respaldada por la experiencia acumulada en numerosos proyectos del grupo de investigación KAOS pero, de acuerdo con este estudio [291], la puesta en práctica de este enfoque en un caso de estudio real está todavía pendiente.

Después de este trabajo, en el ámbito de KAOS se ha llevado a cabo una investigación para proyectar modelos declarativos de metas a requisitos operacionales con la intención de conseguir lo mejor de ambas aproximaciones:

- Operaciones especificadas por pre y postcondiciones y disparadores [183] (S5). Las metas funcionales asignadas a los agentes software deben ser operacionalizadas en las especificaciones de los servicios del software que los agentes deberían proporcionar para satisfacer tales metas, mediante la aplicación de *patrones operacionales* (*operationalization patterns*). Los autores reconocen su limitada experiencia con estos patrones, que está basada principalmente en un conjunto de casos de estudio procedentes de la literatura. Alrajeh *et al.* [24] (S6) afirman posteriormente que este refinamiento de los modelos de metas en requisitos operacionales es una tarea manual, tediosa, sólo parcialmente soportada por los patrones operacionales. Alrajeh *et al.* se plantean este problema proponiendo un enfoque semiautomático basado en comprobación de

modelos (*model checking*) y en aprendizaje inductivo. Se trata de una investigación en fase inicial que se describe a través de un caso de estudio sobre las precondiciones de aprendizaje para los modelos KAOS. Alrajeh *et al.* creen que su método puede ser adaptado para generar otros requisitos operacionales, como disparadores.

- Especificaciones tabulares basadas en eventos para software de control, escritas en el lenguaje SCR [83] (S7). Estas especificaciones constituyen un método bien establecido en el desarrollo de software de control para especificar requisitos operacionales, y proporcionan técnicas y herramientas sofisticadas para el análisis de los modelos del software. De nuevo esta propuesta es ilustrada a través de un ejemplo encontrado en la literatura.

En una conferencia en el congreso RE'04, van Lamsweerde [290] (S8) reflexionaba sobre la investigación y la práctica relacionadas con la IR basada en metas. Una de sus reflexiones se refería a "la necesidad de soporte automatizado adecuado", tras lo cual presentaba su herramienta Objectiver, junto con las lecciones aprendidas y los desafíos relacionados. Objectiver es un conjunto de herramientas comerciales (www.objectiver.com) que soportan el método KAOS. Se trata de un entorno maduro y bien documentado que ha evolucionado a partir del prototipo GRAIL. El conjunto de herramientas de Objectiver hace posible generar documentos de requisitos de forma fuertemente asistida. La estructura de estos documentos procede del grafo de refinamiento de metas y de las plantillas de documentos de requisitos que reflejan estándares específicos de la organización (se incluye también el estándar IEEE 830 [138]). El documento de requisitos contiene un glosario de términos generado a partir del modelo de objetos, anotaciones textuales recuperadas del modelo, y figuras seleccionadas por el usuario a través de *arrastrar-y-soltar* a partir de los submodelos de metas, objetos, agentes y operaciones. Objectiver es el único trabajo encontrado en esta RSL que podemos catalogar como *práctica industrial*.

4.2.5.4 Otras derivaciones de especificaciones de requisitos desde modelos de objetivos

La propuesta conjunta de los grupos de Yu y Mylopoulos y de Du Bois y Dubois [300] (S9) conjuga dos marcos orientados a agentes para la IR: se trata de la especificación de requisitos (fundamentalmente funcionales) en el lenguaje Albert a partir del modelado del negocio basado en metas con i*. Albert es un lenguaje formal que utiliza una lógica temporal de primer orden. Con este enfoque combinado se postula que el proceso de requisitos puede iterar entre los niveles de *especificación* y de *comprensión* hacia una especificación de requisitos: el nivel de especificación o modelado prescribe *lo que* los agentes deben hacer o saber, y el nivel de comprensión o análisis describe *porqué* los agentes se relacionan unos con otros de cierta manera y *porqué* podrían preferir otra configuración de relaciones. El objetivo de este trabajo no es traducir diagramas semiformales i* a especificaciones formales Albert, sino usar i* como una primera notación de modelado para extraer metas de alto nivel antes de convertirlas en requisitos formales de grano más fino. Se trata, por tanto, de un enfoque *integrador* (véase *Modo comb.* en la Tabla 47). Este estudio presenta una contribución inicial en la que se debe trabajar el método para obtener requisitos del sistema, pues no se presenta un algoritmo, reglas ni guías para trasladar un modelo i* en especificaciones Albert. La propuesta se ilustra a través de un ejemplo académico.

GBRAM (*Goal-Based Requirements Analysis Method*) es un método de IR propuesto por Antón y Potts [27] (S10) que permite inferir metas a partir de requisitos informales y después derivar requisitos operacionales más completos a partir de tales metas. Este estudio ha sido incluido en la RSL dado que la última actividad de GRAM relacionada con el refinamiento de metas es "Operacionalizar", que consiste en la traducción de las metas en requisitos operacionales para la especificación final de requisitos. Estos requisitos se especifican por medio de plantillas que contienen (1) una meta refinada; (2) pre y postcondiciones; y (3) escenarios asociados. Nótese que el modo de combinación de este estudio es *integrador* y no *generativo* (Tabla 47). GBRAM no propone, por tanto, algoritmos, reglas o patrones a partir de los cuales derivar los requisitos operacionales, sino que proporciona un conjunto de heurísticas, la utilización secuencial de cuestiones sistemáticas, la relajación de las metas iniciales considerando *obstáculos* (cualquier cosa que puede suceder y que amenaza la consecución de una meta), y la exploración de escenarios. Una contribución interesante de este trabajo es que los documentos de requisitos deben ser documentos *vivos* en el sentido de que deben registrar los requisitos, las cuestiones abiertas que aparecen y desaparecen dinámicamente y los *requisitos de organización* (*organizational requirements*) que contienen información importante para el proceso de IR (por ejemplo, la persona que conoce bien determinados requisitos o la que en última instancia será afectada por una decisión). La salida del proceso completo consiste en un documento de requisitos cuya estructura está basada en las principales áreas funcionales del sistema, cada una de ellas formada por las siguientes subsecciones: (1) Metas; (2) Requisitos funcionales; (3) Requisitos no funcionales; y (4) Requisitos de organización. Una limitación de este estudio es que no considera la relación entre metas y requisitos no funcionales. Este trabajo, conducido por medio de I-A, ha sido validado en distintos estudios que incluyen una aplicación de comercio electrónico.

4.2.5.5 Derivación de requisitos desde modelos del negocio

La generación de requisitos a partir de modelos de procesos del negocio incluye un trabajo de Cox *et al.* [61] (S11) en el que se pretende (1) conocer la aplicabilidad en proyectos reales de los *marcos de problemas* (*problem frames*) propuestos por Michael Jackson; (2) conectar el modelado de procesos del negocio mediante la notación RAD (*Role-Activity Diagram*) con estos marcos de problemas; y finalmente (3) derivar requisitos desde los modelos de procesos. En nuestra opinión, esta *derivación* de requisitos no es sistemática, sino que es uno de los pasos para obtener un marco de problema relacionado con el modelo de procesos e implica la producción de una especificación de requisitos en la cual el ingeniero de requisitos aplica su conocimiento sobre el problema. Pensamos que, a pesar del título de este artículo, la *derivación* de requisitos tiene un papel secundario en el mismo. La investigación es validada con un caso de estudio conducido mediante I-A acerca de un sistema real de comercio electrónico.

En la línea de derivación de requisitos a partir de modelos de procesos del negocio también se encuentra la contribución de Türetken *et al.* [287] (S12), que proponen un patrón para describir en lenguaje natural una parte de los modelos de procesos escritos en la notación eEPC. Este patrón se aplica automáticamente a través de una herramienta y según los autores permite una "importante ganancia de productividad", si bien en los casos de estudio reportados (dos grandes aplicaciones militares) el 40% de los requisitos exigió modificaciones. Se trata, por tanto, de un trabajo en la línea de los de Maiden *et*

al. (Sección 4.2.5.2), si bien se utiliza un solo patrón de requisitos y no se contempla todavía la generación de sentencias condicionales.

4.2.5.6 Derivación de requisitos desde modelos UML

La propuesta de Arlow *et al*. sobre modelado literario (Sección 4.2.5.1) está relacionada con UML pero no proporciona algoritmos, reglas o patrones específicos para derivar requisitos a partir de modelos UML. Dejando aparte casos de uso y escenarios (que se discuten en la próxima Sección 4.2.5.7), sólo se ha encontrado una propuesta en relación con UML.

Meziane *et al.* [215] (S13) introducen otra interesante línea de investigación en la RSL: los sistemas de generación de lenguaje natural. Estos autores proponen el prototipo *GeNLangUML* (*Ge*nerating *N*atural *L*anguage from *UML*), que por medio de una ontología lingüística denominada *WordNet* genera especificaciones en inglés que *parafrasean* diagramas de clases UML 1.5. La intención de Meziane *et al.* es doble: (1) por un lado desean proporcionar a los usuarios en cualquier momento dos *vistas* de la especificación del sistema: UML y lenguaje natural; (2) por otro lado encuentran otra motivación para su trabajo en un proceso de ingeniería inversa durante la etapa de mantenimiento, para permitir transformaciones hacia atrás que permitan a los interesados visualizar en lenguaje natural los cambios en la implementación del sistema. La evolución del sistema se deriva del código fuente, la notación de diseño –UML– y las especificaciones del sistema en lenguaje natural. La generación de texto se ejemplifica por medio de un estudio académico sobre un sistema universitario, donde la especificación generada incluye requisitos como “Una persona es un profesor o un estudiante”, “Un profesor tiene un nombre, una dirección particular, privilegios de aparcamiento, una fecha, un seminario y un seminario supervisado”, “Cero o más profesores imparten cero o más seminarios”, “Una persona vive en una dirección”, “Una dirección tiene una calle, una ciudad, un estado y un código postal”, etc. GeNLangUML extiende el sistema *ModEx (Model Explainer),* un trabajo anterior de Lavoie *et al.* [179].

Meziane *et al.* identifican ciertas debilidades en su propuesta, que es puramente académica. En primer lugar, el enfoque asume ciertas convenciones de nombres extraídas de libros de texto. Estas convenciones podrían cambiar en la práctica industrial de manera que la herramienta debería ser configurada en consecuencia. En segundo lugar, el nivel de abstracción del texto generado es cercano al nivel de abstracción del modelo UML inicial. En tercer lugar, estos autores postulan que un sistema ideal para generar lenguaje natural a partir de modelos orientados a objetos debería incluir diagramas de clase, diagramas de interacción (secuencia y colaboración), diagramas de estado, diagramas de actividades y OCL. En relación con los diagramas de interacción, dudamos de la necesidad de incluir diagramas de colaboración pues pensamos que están relacionados con el diseño más que con los requisitos. Los diagramas de secuencia, sin embargo, pueden utilizarse para describir escenarios de casos de uso. Finalmente, Meziane *et al.* comentan que su trabajo se puede complementar con la investigación de Burke y Johannisson [49] sobre la traducción a lenguaje natural de especificaciones OCL.

4.2.5.7 Casos de uso, escenarios e historias de usuarios

Los casos de uso constituyen una técnica bien conocida para extraer, analizar, especificar y validar requisitos, que dispone de una representación gráfica sencilla y ampliamente extendida, si bien los casos de uso son de naturaleza fundamentalmente textual [174]. Esta sección trata con casos de uso, escenarios, e historias de usuario en combinación con requisitos textuales y documentos. Hemos incluido aquí un conjunto heterogéneo de propuestas cuyo objetivo es distinto pero pensamos que complementario, en el sentido de que puede ser aplicado siguiendo una secuencia lógica de acciones: (1) en primer lugar se discute la investigación de Maiden *et al.* que se dirige a mejorar la completitud de los requisitos a través del análisis de los escenarios del sistema [192, 194, 212]. Este proceso utiliza el modelo de casos de uso existente como un punto de inicio y deriva nuevos escenarios teniendo en cuenta situaciones que no han sido consideradas todavía. Esta derivación de nuevos escenarios produce, por su parte, nuevos requisitos; (2) a continuación se examinan los trabajos de Berenbach [38, 39] y Firesmith [105] que estudian la generación de requisitos en lenguaje natural a partir de modelos de casos de uso. Berenbach se ocupa de la generación de la jerarquía de requisitos mientras que Firesmith propone un patrón con el que derivar el texto de los requisitos. Cabral y Sampaio [50], en contraste, investigan la generación de requisitos en lenguaje formal a partir de los modelos de casos de uso; finalmente (3) Daniels *et al.* [81] y Probasco y Leffingwell [253] tratan con los documentos de requisitos que van a ser desarrollados a partir de casos de uso y escenarios, integrando requisitos tradicionales tipo *El sistema deberá…* con casos de uso. Estos últimos dos trabajos no se ciñen estrictamente a la generación de requisitos textuales a partir de modelos, pero han sido incluidos en la RSL para ejemplificar las guías relacionadas con los contenidos del documento de requisitos que debe ser construido cuando casos de uso y requisitos textuales se combinan en IR.

Durante largo tiempo Maiden *et al.* han investigado el proceso de desarrollo de escenarios y casos de uso con el fin de encontrar requisitos ausentes, en busca de la completitud de la especificación de requisitos. Estos autores han desarrollado una base de datos que contiene una jerarquía de condiciones excepcionales y de error que se utiliza para generar propuestas de caminos alternativos que luego han de ser validados. Estos caminos alternativos se generan a partir de cuestiones *Qué pasaría sí…* (*what-if*) relativas a un curso de eventos normal. La base de datos contiene una lista de 54 clases de conducta y estados anormales, que se puede usar manualmente para cada evento, como una lista de chequeo, o automáticamente través de la herramienta ART-SCENE en el proceso RESCUE. Esta taxonomía general se ha extendido con conocimiento de varios dominios de aplicación. Una versión inicial de esta herramienta (denominada CREWS-SAVRE) se presenta en detalle en Maiden *et al.* [192] (S14) a través del análisis retrospectivo de los escenarios de una parte del Servicio de Ambulancias de Londres. Mavin y Maiden [212] (S15) estudian después qué tipos de escenarios y qué técnicas de recorrido (*walkthrough*) son más efectivos para descubrir requisitos, y lo hacen a través de dos casos de estudio: (1) un simulador en un dominio de barcos de guerra en BAE SYSTEMS; y (2) un Asistente de Resolución de Conflictos (llamado CORA-2) en el dominio de gestión de tráfico aéreo en Eurocontrol. Mavin y Maiden proporcionan guías para el descubrimiento de requisitos basado en escenarios, y curiosamente sugieren que los recorridos sistemáticos de escenarios sencillos que no contienen demasiado conocimiento del dominio son más efectivos para descubrir requisitos del sistema. Construyendo sobre este trabajo, Maiden y Robertson [194]

(S16) llevan a cabo un análisis retrospectivo de experiencias anteriores para investigar cómo los casos de uso y los escenarios fueron desarrollados durante la aplicación del proceso basado en escenarios RESCUE en DMAN, un sistema de gestión del tráfico aéreo para el Servicio de Tráfico Aéreo Nacional del Reino Unido. Maiden y Robertson también estudian la conexión entre escenarios y requisitos en lenguaje natural especificados con la plantilla VOLERE.

Esta influyente investigación de Maiden *et al.* sigue todavía extendiéndose. En este párrafo se recoge trabajo adicional que complementa el análisis de los trabajos anteriores seleccionados en la RSL. Este trabajo consiste de (1) una extensión de ART-SCENE para incluir ricos escenarios multimedia [301], que sirve para estudiar la mejora del descubrimiento de requisitos por medio de otras formas de escenarios, como simulaciones visuales de los agentes en el dominio, que se presentan a los interesados junto con escenarios textuales en ART-SCENE; y (2) la herramienta *Mobile Scenario Presenter* (MSP) [270], que es una extensión de ART-SCENE que sirve para investigar el uso de PDAs (*Personal Digital Assistants*) para soportar in situ los recorridos de escenarios ART-SCENE. Maiden *et al.* pretenden estudiar entonces [193] si las simulaciones visuales de escenarios y los recorridos de escenarios en el contexto de trabajo pueden evocar requisitos que podrían no haber sido descubiertos mediante los recorridos de escenarios con ART-SCENE.

Berenbach [38] (S17) ha diseñado un algoritmo para la extracción automática de requisitos a partir de diagramas de casos de uso. En nuestra opinión, este autor utiliza los diagramas de casos de uso de una forma particular, como *diagramas conceptuales* o mejor como *diagramas de características* [160] del dominio bajo estudio. A través del algoritmo propuesto los resultados se organizan en forma de árbol de requisitos: los denominados *casos de uso abstractos* se transforman en lo que denomina *características* y *subcaracterísticas*, mientras que sus *casos de uso concretos* se transforman en *requisitos detallados* para los cuales se pueden crear tareas del proyecto y generar casos de prueba. Este enfoque se concentra en la generación de una jerarquía de requisitos más que en el propio texto de los requisitos, y ha sido probado en modelos complejos procedentes de empresas que trabajan con Siemens. En [39] (S18), Berenbach refina estos resultados para tratar la síntesis de requisitos textuales y la denominada *IR dirigida por modelos*. Esta síntesis resulta en un enfoque de IR que integra sin costuras casos de uso, características y requisitos. En este trabajo Berenbach propone un conjunto de buenas prácticas y recomendaciones, entre las que cita la automatización de la generación de documentación de casos de uso y requisitos. La experiencia en la industria de Berenbach le lleva a afirmar que la propuesta recogida en [39] conlleva entre un tercio y un medio del tiempo de un enfoque tradicional.

La intención de Firesmith [105] (S19) es la derivación de un conjunto completo de requisitos no ambiguos y verificables que proceden de historias de usuario, escenarios y casos de uso “incompletos y vagos”. Con este fin, Firesmith propone una plantilla estándar para derivar requisitos textuales a partir de las interacciones de los caminos de los casos de uso, en la forma: “Si un *disparador* ocurre cuando ciertas *precondiciones* se dan, entonces el sistema deberá realizar un conjunto de *acciones* y se satisfacerá un conjunto requerido de *postcondiciones*”. Esta plantilla se aplica manualmente a la especificación de casos de uso, y da lugar a una especificación de requisitos en lenguaje natural que los interesados pueden comprender y validar. Firesmith postula que el esfuerzo extra requerido para desarrollar una especificación de requisitos es pronto

recuperado gracias al esfuerzo ahorrado durante el resto de las actividades de desarrollo de software. Los requisitos textuales generados se especifican en lenguaje natural con el fin de permitir a los interesados su validación. Sin embargo, estos requisitos son largos y complejos y pueden no ser fácilmente legibles. Este problema, que Firesmith asocia con "la inevitable complejidad de los requisitos completos", puede ser mitigada estructurando las sentencias en sus partes constituyentes. El autor muestra la aplicación de la propuesta a través de un clásico ejemplo académico sobre un cajero automático.

La investigación de Cabral y Sampaio [50] (S20) introduce la generación de especificaciones formales a partir de modelos de casos de uso. Estos autores proponen una estrategia a través de la cual traducir automáticamente casos de uso escritos en un subconjunto del inglés (denominado CNL, *Controlled Natural Language*) en una especificación en álgebra de procesos en CSP. CNL incluye una ontología que describe las entidades en el dominio de aplicación. Los casos de uso se especifican por medio de dos niveles de plantillas: (1) *casos de uso de la vista de usuario* (*user view use cases*), que diseñan cómo los actores interactúan con el sistema; y (2) *casos de uso de la vista de componentes* (*component view use cases*), que especifican la conducta del sistema a partir de la interacción del usuario con los componentes del mismo. Este enfoque puede ser útil en aquellos dominios en los cuales la consistencia de los requisitos es especialmente importante, como en las telecomunicaciones. Este trabajo surge específicamente de un proyecto de colaboración con Motorola. La especificación formal resultante puede ser utilizada para generar automáticamente casos de prueba. Esta especificación formal no es legible para los interesados, pero no pensamos que esto sea un problema pues los interesados pueden validar el modelo de casos de uso inicial. El uso de un lenguaje controlado también ayuda a evitar ambigüedad en las plantillas de casos de uso. Esta propuesta es soportada completamente por un conjunto de herramientas que consiste de (1) un *plugin* para Ms Word con el fin de editar especificaciones de casos de uso para la gramática de CNL; (2) un traductor de casos de uso CNL a CSP; y (3) FDR, un comprobador de modelos CSP, para verificar el refinamiento entre las vistas de usuario y de componentes.

Daniels *et al.* [81] (S21) proponen un método práctico con el cual integrar casos de uso y requisitos tradiciones *El sistema deberá…* Para estos autores los casos de uso no son requisitos, sino un vehículo para descubrir requisitos. Proponen el uso de un *Segmento de Requisitos Funcionales* (*Functional Requirements Segment*) en la sección de *Requisitos Específicos* de la plantilla de casos de uso, en el cual se incluyen los requisitos *El sistema deberá…* que el ingeniero de requisitos extrae manualmente a partir de las sentencias del escenario (nótese que una sentencia en un caso de uso puede contener múltiples requisitos funcionales). Los requisitos textuales retienen su contexto gracias a las trazas a los eventos o secuencias de eventos del caso de uso a partir de los cuales han sido derivados. Cuando los requisitos se documentan como un conjunto de sentencias *El sistema deberá…*, sin contexto, es difícil comprenderlos e interpretar completamente su intención y sus dependencias. La *Especificación de Requisitos Suplementaria* contiene los requisitos que no se ajustan completamente a un solo caso de uso. Si se debe desarrollar una especificación de requisitos tradicional entonces el ingeniero de requisitos debe copiar y pegar manualmente los requisitos documentados en la sección Requisitos Específicos de cada caso de uso, junto con los requisitos en la Especificación de Requisitos Suplementaria. El enfoque se ilustra a través de un ejemplo académico sobre una línea de productos software en un microondas.

Probasco y Leffingwell [253] (S22) muestran una propuesta de Rational Software, similar a la de Daniels *et al.*, para combinar casos de uso y especificaciones tradicionales de requisitos a través de una construcción sencilla que denominan *Paquete de Especificación de Requisitos del Software* (*SRS Package*). Este paquete contiene el conjunto completo de requisitos software para un sistema, que pueden estar contenidos (1) en un documento individual; (2) en múltiples documentos de requisitos; (3) en un repositorio de requisitos (que consiste del texto de los requisitos, sus atributos y trazabilidad); y (4) en especificaciones de casos de uso y diagramas de casos de uso. Este trabajo de Probasco y Leffingwell [253] es solo un corto *white paper* y no se menciona ningún procedimiento concreto de validación.

4.2.5.8 Derivación de requisitos desde modelos de interfaz de usuario

Por último, pero no por ello menos importante, se encuentra un grupo de trabajos que generan documentación textual de requisitos a partir de modelos de la interfaz de usuario. Se trata de requisitos cercanos al diseño del sistema. Jungmayr y Stumpe [157] (S23) proponen *modelos de uso extendidos* que constan de tres submodelos: *modelo de escenarios*, *modelo de acciones* y *modelo de interfaz de usuario*: (1) el modelo de escenarios describe la semántica del modelo de uso en términos de metas que pueden ser conseguidas utilizando el software, tareas que deben ser realizadas para conseguir tales metas, y soluciones, que son escenarios que muestran cómo usar el software para resolver una tarea particular; (2) el modelo de acciones consiste en una máquina de estados que define todas las posibles secuencias de entradas de usuario (acciones) y engloba la información capturada por un modelo de uso convencional; y finalmente (3) el modelo de interfaz de usuario describe la interfaz de usuario del software y los elementos de la misma que pueden registrar entradas del usuario. A partir de este modelo de uso extendido se puede generar automáticamente un documento HTML que estructura la información ya introducida y que sirve como documentación del usuario. La construcción de los modelos de uso extendido requiere un esfuerzo considerable, pero los autores afirman que también lo requiere la construcción de los modelos de uso convencionales. La estructura de los documentos de salida se muestra en detalle en [157]. Los datos de los documentos de salida están relacionados por medio de patrones de texto predefinidos. Este trabajo ha sido evaluado en una aplicación de comunicaciones para la recuperación de referencias bibliográficas de bases de datos comerciales. El ejemplo mostrado es simple pero está bien evaluado.

Un trabajo antiguo de Smith [273] (S24) en MITRE Corporation sirve para presentar un método para generar requisitos funcionales relativos a la interfaz de usuario a partir de cuestionarios (*check-list*) sobre la misma, en los cuales el analista marca si una característica concreta de la interfaz es *requerida*, *útil* o *no necesaria*. Para generar los requisitos en lenguaje natural se utiliza la denominada *prosa en patrones* (*patterned prose*), que sirve para parafrasear los cuestionarios. En esencia la prosa en patrones consiste en una jerarquía de sentencias, frases y palabras, y sus conectores lógicos, organizados (y numerados) de forma que se correspondan con la estructura de los cuestionarios que se usan para extraerlos. Esta aproximación está muy centrada en el diseño de la interfaz de usuario. En nuestra opinión, el enfoque de Smith está muy cercano al diseño de la interfaz de usuario: parece más a un nivel de especificación real (físico) que *esencial* (lógico). Las aplicaciones de hoy día tienen una GUI mucho más compleja que la ilustrada en los ejemplos de este trabajo, y por tanto pensamos que sería

necesario un extenso trabajo de análisis del dominio para redefinir los cuestionarios con vistas a cubrir las aplicaciones actuales. Sin embargo, la idea de prosa en patrones nos ha parecido interesante. Una implementación sobre UNIX estaba disponible, aunque en este trabajo no se reportan procedimientos de validación.

4.2.5.9 Limitaciones de esta revisión sistemática de la literatura

Como se ha mencionado previamente, las búsquedas en esta RSL se han definido utilizando términos muy comunes de ingeniería del software, como *modelo*, *requisito* y *especificación*, términos que podríamos decir que están *sobrecargados* con numerosos significados. En un campo de estudio heterogéneo como el definido en la Sección 4.2.1.1 ha sido necesario utilizar estos términos de uso general para buscar los estudios a analizar. Los términos utilizados para construir la cadena de búsqueda (Sección 4.2.1.2) tienen más sinónimos y algunos de estos términos son homónimos. Por consiguiente no podemos garantizar que todo el trabajo relacionado con el ámbito de esta RSL haya sido encontrado por medio de las consultas realizadas. Sin embargo, después de comprobar las referencias a los estudios seleccionados en esta RSL, algunos de los cuales han sido publicados en revistas o conferencias importantes, creemos que hemos analizado los contenidos de una muestra ilustrativa de este campo, si bien alguna importante publicación (la de Meziane *et al.* [215], S13) ha sido incluida directamente en la revisión como resultado de nuestro conocimiento de este campo, pues no había aparecido en los resultados de las búsquedas.

Como se indica en la Sección 4.2.2, hemos introducido dos desviaciones del protocolo con el objetivo de mejorar la completitud del conjunto de publicaciones seleccionadas. Pensamos que la discusión es así más completa, aunque la repetibilidad de la RSL se ve comprometida.

Consideramos *Herramienta* y *Validación* como parte de los criterios de evaluación de la calidad (Sección 4.2.1.5), pero estos elementos son difíciles de cuantificar con precisión. La mayor parte del soporte automatizado involucrado en los estudios seleccionados no es accesible y por tanto es difícil valorar con precisión el nivel de madurez de las herramientas. Más aún, la corrección de los procedimientos de validación no puede ser evaluada con precisión a partir de la lectura de los trabajos.

4.2.6 Derivación de requisitos del producto en líneas de productos software

Durante el análisis de los estudios seleccionados concluimos que una de las debilidades de esta RSL es que ningún trabajo se dirigía específicamente a las líneas de productos software (LPS). Conocíamos una herramienta comercial para LPS, GEARS [11], que automáticamente deriva una especificación de requisitos del software que describe en lenguaje natural las decisiones adoptadas en la instanciación de los puntos de variación de la LPS y las características obligatorias de los productos de la misma. Intuitivamente, pensábamos que las LPS constituyen un dominio en el cual la derivación de requisitos a partir de modelos puede jugar un papel interesante: la derivación del producto en LPS implica la navegación a través de un *espacio de decisión* (como por ejemplo, un modelo de características [160] o un modelo independiente de variabilidad [249]), que resuelve un conjunto de puntos de variación para producir la especificación de un producto en la

línea de productos. Pensábamos que la especificación de este producto en lenguaje natural por medio de un documento de requisitos podría ayudar a clientes a comprender la configuración del producto que adquieren. Estas reflexiones nos llevan a extender la RSL anterior realizando una nueva RSL sobre la derivación de requisitos del producto en LPS. Esta nueva RSL se describe por completo en esta sección.

Los procesos detallados para guiar la derivación de los requisitos no están suficientemente estudiados y generalmente no se manejan por parte de las herramientas de LPS [87]. En el paradigma de LPS existe un consenso sobre la necesidad actual de prestar más atención al proceso de derivación del producto (véase, por ejemplo, el prefacio de Käkölä y Dueñas en [159]). En esta línea Bühne *et al.* [47] presentan una visión general de la investigación sobre la derivación de la especificación del producto en LPS. Desde un punto de vista general, estos autores examinan el desarrollo de especificaciones de requisitos de la aplicación, prestando especial atención al tratamiento de los nuevos requisitos de los interesados que no están incluidos en la LPS (denominados *deltas*), pero no abordan el tema específico de esta RSL. Esta Sección 4.2.6, por el contrario, se dedica expresamente a las propuestas que consideran explícitamente la derivación de requisitos a partir de modelos de variabilidad de la LPS durante la IR de la aplicación. De esta forma, aparece una nueva cuestión:

> RQ3. ¿Qué enfoques tienen en cuenta la derivación de requisitos a partir de modelos de LPS?

Para responder esta cuestión han sido utilizadas las cadenas de búsqueda que se muestran en la Tabla 48. Las búsquedas han sido realizadas en Google Scholar pues ha sido la fuente que ha proporcionado más resultados en la RSL general. Los criterios de inclusión y exclusión son los mismos que los de la RSL general, aunque como es natural restringidos al ámbito del dominio de las LPS. Los procedimientos de recolección y análisis de datos también han sido los mismos. De esta forma se han seleccionado cinco trabajos (ver Tabla 49). Siguiendo el estudio S25 en la Tabla 49 hemos considerado interesante incluir en la selección el estudio S26 (se produce así una desviación del protocolo), y así finalmente se han seleccionado seis estudios, que se discuten en el resto de esta sección. El *modo de combinación* de los seis estudios es *generador* y su ámbito es *requisito*. Ningún estudio trata con la generación de documentos de requisitos. Los modelos iniciales son modelos de características y modelos de variabilidad (cuatro y dos estudios, respectivamente), mientras que los modelos destino vienen dados por el lenguaje natural (cuatro de las propuestas) y las notaciones formales (tres estudios). Nótese que S27 se dirige tanto a notación formal como a lenguaje natural.

En relación con la derivación de requisitos del producto en LPS, un trabajo pionero es el de Hein *et al.* [131] (S25 en la Tabla 49), que afirma que el conocimiento del dominio debe ser formalizado para permitir un proceso parcialmente automatizado de derivación de requisitos a partir de un modelo de características. Hein *et al.* se centran en un *modelo de características* (que extiende el de FODA [160]) y un *modelo de requisitos* (que engloba los textos de los requisitos y las definiciones de los parámetros para un producto o para el dominio). Un requisito puede especificar variabilidad a través de parámetros. Un requisito que contiene parámetros se considera un *requisito plantilla*. Las características se modelan como los nodos de un árbol. Básicamente, cada nodo se corresponde con un parámetro. El árbol de características por tanto describe los posibles

valores que se pueden asignar a un parámetro. Hein *et al.* afirman que las herramientas CARE tradicionales han sido diseñadas para describir productos individuales, no familias de productos, y que por tanto estas herramientas no se pueden aplicar a las LPS. Hein *et al.* enuncian varios problemas que han encontrado a través de su experiencia en gestionar los artefactos del análisis del dominio mediante herramientas CARE, especialmente con QSS DOORS. Posteriormente, Hein *et al.* imponen nueve requisitos sobre una herramienta CARE para LPS, entre los que incluyen la "generación automática de textos de requisitos específicos". La generación de especificaciones de requisitos puede ser automatizada parcialmente con el uso de plantillas instanciadas con distintas variantes, lo cual estos autores consideran muy útil en un contexto de LPS, en el cual distintos productos y variaciones de productos deben ser derivados a partir de descripciones abstractas del dominio. Hein *et al.* aseguran que una representación textual es más adecuada para la discusión con expertos del dominio que modelos más formales en los cuales la información se encuentra dispersa. Estos autores aseguran también que el problema de la derivación del producto en una LPS se puede gestionar con sistemas basados en conocimiento, y especialmente sistemas de configuración, como el de Konwerk [268] (S26). Estos resultados se han consolidado y validado en un experimento industrial en Bosch, en un dominio de supervisión del entorno de automóviles (*Car Periphery Supervision*, CPS).

Cadenas de búsqueda	Estudios encontrados	Estudios candidatos	Estudios seleccionados
+"requirements derivation" +"software product lines",	14	3	3
+"deriving requirements" +"software product lines"	6	2	2
+"requirements integration" +"software product lines"	4	0	0
+"integrating requirements" +"software product lines"	14	0	0

Tabla 48 Búsqueda de enfoques relacionados con líneas de productos en Google Scholar (cuestión de la investigación RQ3)

El trabajo anterior de Hein *et al.* indica que la representación del conocimiento del dominio debe estar formalizada para permitir un proceso de derivación parcialmente automatizado a partir del modelo de características. Estos autores también manifiestan que las herramientas CARE no proporcionan guías para el cliente en relación con la configuración del producto en el proceso de instanciación de una LPS. Pensamos que un trabajo corto de Rabiser *et al.* [255] (S27) también se puede clasificar en este grupo. Estos autores presentan una herramienta, denominada DOPLER (*Decision-Oriented Product Line Engineering for effective Reuse*), que soporta el modelado de la variabilidad y la configuración del producto en LPS. DOPLER incluye un *asistente para la configuración* (*ConfigurationWizard*) para adoptar decisiones durante la derivación del producto, permitiendo la adaptación del producto, la captura de requisitos, y la generación de la configuración. Desde un punto de vista similar al de Hein *et al.*, Rabiser *et al.* afirman que para explotar completamente los beneficios de las LPS es esencial hacer los modelos de LPS accesibles a interesados no técnicos como comerciales o clientes, que toman importantes decisiones durante la derivación del producto. Las decisiones bajo la responsabilidad de un cliente se listan como cuestiones en lenguaje natural respondidas sí o no. Un grafo y una vista en árbol muestran las relaciones entre las decisiones para soportar la navegación en el espacio de decisión proporcionado por el modelo de variabilidad. Los nuevos requisitos que no son todavía

contemplados por la LPS se consideran también, y son descritos utilizando la plantilla VOLERE. Así la configuración del producto generado a partir del modelo de variabilidad consiste de una representación tabular formal de las características seleccionadas junto con las plantillas de los nuevos requisitos textuales. El asistente para la configuración se puede utilizar también para lanzar aplicaciones de simulación basadas en los assets seleccionados. Las referencias de este trabajo nos permiten determinar que esta investigación se lleva a cabo para soportar un escenario típico en los procesos de derivación y venta del producto, en relación con una LPS cuyo objetivo es la automatización de la fundición continua de acero en plantas de SIEMENS VAI.

Tavakoli y Reiser [279] (S28) proponen una biblioteca de requisitos (*requirements library*) basada en un nuevo modelo para gestionar la variabilidad y las partes comunes de familias de productos en el nivel de los requisitos. Este estudio surge de una iniciativa de mejora del proceso en DaimlerChrysler, en el dominio de la automoción, donde la variabilidad es especialmente compleja debido a que están implicadas *líneas de productos jerárquicas* (*hierarchical product lines*): distintos modelos de vehículos contienen unidades de control electrónicas que son, a su vez, líneas de productos. Los requisitos juegan un papel vital en esta industria, pues el resto de artefactos del software tales como la arquitectura o el código se desarrollan por parte de proveedores. La eficiencia en la obtención de especificaciones para ciclos de desarrollo cortos es de creciente importancia. En este enfoque, la derivación de especificaciones del producto a partir de una biblioteca de requisitos significa poder reducir un paso la variabilidad. La reutilización de requisitos en la biblioteca de requisitos se dirige por medio de la selección de características, que a su vez preselecciona requisitos en la base de datos. Por tanto, la principal tarea del ingeniero de requisitos pasa de ser el desarrollo de especificaciones de requisitos a la *derivación* de especificaciones de requisitos (que incluye la derivación de la definición de la variabilidad). El soporte automatizado ha sido desarrollado a través de la herramienta CARE comercial DOORS [9] por medio de su lenguaje *script dxl* (*DOORS eXtension Language*). Para estos autores la aplicabilidad de la propuesta en una herramienta es un paso importante hacia su implantación práctica. Aunque parece que el estudio procede de un conocimiento adecuado del dominio de la automoción, no se informa de que el enfoque haya sido aplicado en la práctica.

La investigación en curso de Djebbi y Salinesi sobre RED-PL (*Requirements Elicitation & Derivation for Product Lines*) [85] (S29) está concebida para definir un método para la derivación de requisitos del producto en LPS. De forma distinta a las propuestas anteriores, RED-PL no consiste en la selección de la configuración de un producto a partir de un modelo de variabilidad de la LPS con vistas a recuperar los requisitos que especifican el producto a construir. En contraste, el enfoque de RED-PL consiste en (1) extraer los requisitos de los clientes (a los que se deja libertad para escoger la manera en la cual desean especificar sus requisitos); (2) comparar los requisitos de los clientes con los requisitos de la LPS, generando un conjunto de requisitos que se quieren y que no se quieren; y (3) derivar una colección consistente de requisitos que es óptima para un conjunto de clientes y de restricciones de la organización (por ejemplo, ingresos, coste y recursos, tiempo). Para comparar las necesidades de los clientes, que son expresadas textualmente, con los requisitos de la LPS, se usa una aplicación para resolver restricciones (*constraint solver*) que utiliza técnicas de análisis de similitud (*similarity analysis techniques*). Actualmente los requisitos de la LPS son especificados como

ID	Autor/es [ref.] (fuente)	Año	Modo comb.		Ámbito		Modelo inicial	Texto generado		Método	Herramienta	Validación
			Gen.	Int.	Req.	Doc.		LF	LN			
Sección 4.2.6. Derivación de requisitos del producto en líneas de productos software												
S25	Hein *et al.* [131] (gs)	2000	■	□	■	□	Modelo de características	□	■	No específico	QSS DOORS	(CEI) Supervisión de la carrocería del automóvil en Bosch
S26	Schlick y Hein [268] (Citado por S25)	2000	■	□	■	□	Modelo de características	□	■	No específico	Sistema de configuracion Konwerk	(CEI) Supervisión de la carrocería del automóvil en Bosch
S27	Rabiser *et al.* [255] (gs)	2007	■	□	■	□	Modelo de variabilidad	■	■ Volere	No específico	DOPLER	(CEI) Línea de productos CL2 SPL para fundición continua del acero en SIEMENS VAI
S28	Tavakoli y Reiser [279] (gs)	2007	■	□	■	□	Modelo de variabilidad y biblioteca de requisitos	□	■	No específico	Basada en DOORS a través de un plugin dxl	(CEI) Unidades de control electrónico en DaimlerChrysler
S29	Djebbi y Salinesi [85] (gs)	2007	■	□	■	□	Modelo de características (u otro modelo de análisis de líneas de producto)	■	□	RED-PL	Ms Excel y Ms Excel Solver	(CEI) Analizadores de sangre en Stago Instruments
S30	Djebbi *et al.* [86] (gs)	2007	■	□	■	□	Modelo de características (u otro modelo de análisis de líneas de producto)	■	□	RED-PL	Prototipo en desarrollo basado en GNU-Prolog Solver	(CEI) Analizadores de sangre en Stago Instruments

Leyenda: *fuente*: acm, ieee, sd (ScienceDirect), mpress (MetaPress), wi (Wiley Interscience), gs (Google Scholar), prev.RSL (línea base previa a la RSL), citado por [ref.];
Modo comb. (Modo de combinación): Gen: Generador / Int: Integrador; *Ámbito*: Req: Requisito / Doc: Documento
Texto generado: LF: Lenguaje Formal / LN: Lenguaje Natural;
Validación: (CEA) Caso de Estudio Académico / (CEI) Caso de Estudio Industrial / (PI) Práctica Industrial; I-A: Investigación en Acción

Tabla 49 Estudios de la revisión sistemática considerando RQ3

diagramas de características, aunque este enfoque se puede aplicar a otros lenguajes de modelado de LPS (como por ejemplo casos de uso, metas, UML, aspectos). En [85], Djebbi y Salinesi utilizan programación linear entera (*Integer Linear Programming*, ILP) para el análisis de similitud. En este trabajo los autores utilizan Ms Excel para aplicar ILP a un caso de estudio sobre analizadores de sangre desarrollado en la empresa STAGO Instruments. Algunas dificultades que se observan al aplicar el método son las siguientes: (1) el experimento ha mostrado que ILP no podría ser utilizado adecuadamente donde se tengan que especificar requisitos complejos. La comparación era difícil debido a una falta de precisión en la formulación de los requisitos del cliente. Las dificultades se encontraron no sólo con respecto a la terminología, sino también con respecto a diferencias conceptuales entre los requisitos del cliente y los requisitos de la LPS (diferentes niveles de abstracción, distintas vistas); y (2) ILP también tiene problemas de escalabilidad. Djebbi *et al.* [86] (S30) es esencialmente un resumen de [85] en el cual se utiliza otra técnica de resolución de problemas, programación de restricciones (*constraint programming*), que exhibe mejores resultados en el caso de estudio. Los autores informan del desarrollo en curso de un prototipo de herramienta vía GNU-Prolog Solver. Aunque los autores no definen explícitamente el término *requisito*, parece que un requisito en este contexto puede ser asimilado a una sentencia sobre la configuración de los productos de la LPS. Estos trabajos no se dirigen específicamente a la generación de requisitos en lenguaje natural. Sin embargo, estas propuestas se han analizado ya que introducen un *modus operandi* distinto y porque su salida, un modelo de características instanciado, que consideramos una notación formal, puede ser fácilmente presentada en modo textual.

4.2.7 Conclusiones de la revisión sistemática

En esta RSL se ha revisado exhaustivamente y se ha sintetizado la literatura sobre la generación de requisitos textuales y de documentos de requisitos a partir de modelos del negocio o del software. Se puede objetar que se trata de un tema muy concreto, con poco protagonismo dentro de la investigación en IR. Sin embargo, aunque la generación de requisitos y documentos desde modelos de ingeniería del software ha recibido relativamente poca atención en la investigación (30 trabajos seleccionados en la RSL), creemos haber encontrado en la literatura estudios bien fundamentados que corroboran el interés de esta línea de trabajo. Más aún, dado que 17 de los 30 trabajos seleccionados han sido publicados en los últimos cinco años, pensamos que es razonable afirmar que el interés en este tema, aunque no es general, se está incrementando, especialmente en el dominio de las LPS.

En nuestra opinión, el modelado literario –que pensamos que en sentido amplio puede englobar todo el trabajo seleccionado en esta RSL– es una idea excelente que parece haber tenido escasa relevancia en la práctica. La esencia de esta idea se puede encontrar también en un área relacionada como es la gestión del conocimiento. Por ejemplo, Eriksson [95] afirma que “desafortunadamente, existe un salto sorprendentemente grande entre el conocimiento modelado con ontologías y el texto que documenta ese mismo conocimiento”, y propone una vía para combinar documentos y ontologías, “que permita a los usuarios acceder al conocimiento de múltiples formas”. Pensamos que es interesante hacer un esfuerzo mayor en la generación de requisitos y documentación textual a partir de modelos, en particular en modelado literario y LPS, de manera que (1) esta estrategia se pueda validar empíricamente en un mayor número de entornos industriales; (2) se amplíe el conjunto de técnicas a partir de las cuales los requisitos y

los documentos de requisitos puedan ser generados automáticamente o de forma fuertemente asistida; (3) los modelos de procesos de IR se refinen para tener en cuenta el uso de modelado literario; y (4) se desarrolle el soporte de herramientas CARE comerciales. Creemos que sin un adecuado soporte automatizado esta vía no es verdaderamente aplicable en la práctica, especialmente en el contexto de desarrollos ágiles.

Nos parece que tanto investigadores como profesionales se pueden beneficiar de una mejora en la legibilidad de los modelos de ingeniería del software, haciendo estos modelos accesibles a un abanico de interesados más amplio, facilitando su validación y mejorando su facilidad de uso. En relación con los profesionales, uno de los desafíos que el mapa tecnológico del consorcio industrial ITEA identifica en relación con la dirección del proceso dirigido por los requisitos es precisamente que se requieren vistas especiales que proporcionen a cada interesado una visión adecuada de los requisitos y de cómo son desarrollados [151]. Además, la calidad de la documentación generada por medio de un enfoque de modelado literario puede servir a los profesionales para mejorar la visión de los documentos de requisitos como contrato entre clientes y desarrolladores. Pensamos también que los desarrolladores de tecnología CASE pueden añadir valor a sus herramientas mediante la inclusión de mecanismos de modelado literario. Cuando examina la evolución de la ingeniería del software en las últimas dos décadas, Sommerville [275] concluye que cuando se mira más allá de la tecnología, hacia los procesos fundamentales, desgraciadamente muchas cosas permanecen igual. Una de las cuestiones que Sommerville menciona es precisamente que las herramientas CASE son todavía esencialmente editores de diagramas con alguna funcionalidad de comprobación de modelos y de generación de código.

A partir de los resultados de la RSL postulamos cinco cuestiones clave que deberían ser soportadas por una herramienta CARE que integre sin costuras modelos gráficos y textuales del sistema y/o del software. En términos de la taxonomía definida en la Sección 4.2.3, la herramienta debería ser *generadora* en cuanto a requisitos y a documentos. Además:

1. La herramienta debería permitir la generación automática o fuertemente asistida de documentos de requisitos que integren modelos (del negocio y del software) y requisitos textuales escritos en lenguaje natural, asegurando trazabilidad bidireccional entre modelos y requisitos textuales.
2. Para facilitar la comprensión por parte del ingeniero de requisitos, la estructura de la documentación de requisitos debería seguir la estructura de los modelos: los apartados de la documentación de requisitos deberían evocar de alguna manera la estructura de los modelos. Se podrían promover redundancias para facilitar la comprensión y/o modificación de la documentación, pero deberían estar controladas por la herramienta.
3. Una vez generada, la documentación de requisitos debería quedar *sincronizada* con los modelos, de manera que si cambia un elemento en cualquiera de las dos *vistas* del sistema (la *vista de modelos* o la *vista de texto*), la herramienta debería propagar los cambios necesarios en los elementos relacionados de la otra vista. Además la generación de requisitos se debería ligar con un desarrollo iterativo e incremental, de forma que la generación de requisitos no debería afectar necesariamente a todo el modelo, y las sucesivas generaciones no deberían

sobreescribir los cambios realizados directamente en la documentación. Se hace necesario por tanto un mecanismo de control de versiones. Si un cambio en la vista textual no se corresponde con ningún elemento de los modelos se deberá registrar la aparición de un *delta* que deberá ser registrado y controlado.

4. Como extensión de la cuestión clave 2, la herramienta debería permitir la particularización de la documentación atendiendo al destinatario o al uso de la misma. En particular, la vista textual no sería necesariamente única: podría existir una *vista contrato con el cliente*, una vista *gerente*, una *vista analista*, una *vista desarrollador*, una *vista de pruebas*, etc.

Existen numerosas herramientas comerciales que tienen relaciones con el campo de conocimiento revisado en esta RSL. Por ejemplo, herramientas CARE comerciales bien conocidas, como Requisite Pro [19], DOORS [9], y Caliber-RM [6], están integradas con las herramientas de modelado en UML de la *suite* para permitir la sincronización de modelos de casos de uso y diagramas. Se podría lleva a cabo, por tanto, un trabajo futuro para revisar el mercado de herramientas con la intención de analizar la combinación entre modelos y requisitos en herramientas CARE comerciales.

4.3 Una propuesta de aplanamiento de los modelos de análisis de SIRENspl

4.3.1 Introducción

En este apartado se trata de definir una correspondencia α : DOMMOD $\rightarrow$ REQSPEC, donde:

- *DOMMOD* es el conjunto de todos los posibles modelos de análisis del dominio de líneas de productos realizados con el marco definido en SIRENspl (Capítulo 3). DOMMOD = $DOMMOD_{N1} \cup DOMMOD_{N2}$, donde $DOMMOD_{N1}$ es el conjunto de todos los posibles modelos de líneas de productos, con sus elementos comunes y variables, y $DOMMOD_{N2}$ es el conjunto de todos los productos instanciados dentro de una línea de productos de $DOMMOD_{N1}$. Más concretamente:

    - $DOMMOD_{N1}$ se basa en (1) un modelo de características; (2) un modelo de casos de uso genéricos; (3) una descripción del vocabulario del dominio a través de un modelo conceptual; (4) plantillas de atributos de calidad; y (5) comandos reutilizables. Las características se pueden asociar a (1) casos de uso y/o de pasos dentro de casos de uso; (2) requisitos textuales; (3) mecanismos y estrategias de plantillas de atributos de calidad; y (4) comandos del sistema. Pasos de casos de uso y mecanismos y estrategias en plantillas de atributos de calidad también se pueden asociar a descripciones de comandos del sistema.

- *$DOMMOD_{N2}$* se basa en (1) un modelo de características instanciado, en el cual algunas características están ligadas a (2) casos de uso (concretos o genéricos instanciados) y/o pasos dentro de casos de uso (concretos o parametrizados instanciados); (3) requisitos textuales (concretos o parametrizados instanciados); (4) mecanismos y estrategias de plantillas de atributos de calidad; y (5) comandos reutilizados.

DOMMOD se describe formalmente en la Sección 4.3.3 a través de metamodelado con MOF.

- *REQSPEC* es el conjunto de todas las posibles especificaciones de requisitos de líneas de productos y de productos concretos de una línea de productos compatibles con SIREN (Capítulo 2). Al ser compatibles con SIREN, los requisitos se consideran especificados en un formato fundamentalmente textual, en lenguaje natural, lo cual no excluye el uso puntual de requisitos en formatos complementarios (por ejemplo, tablas o diagramas). Sea *rs* ∈ REQSPEC, entonces *rs* es una especificación de requisitos con un formato abstracto de documento. Posteriormente, este formato abstracto de documento se podría traducir a formatos concretos de documentación de requisitos, como una especificación de requisitos del sistema siguiendo el ejemplo propuesto en el IEEE 1233 [139] y/o una especificación de requisitos del software en formato IEEE 830 [138] o Volere [265]. REQSPEC se describe formalmente en la Sección 4.3.4 a través de metamodelado con MOF.
- Sea *dm* ∈ DOMMOD, entonces tenemos que α (*dm*) = *rs*, donde *rs* ∈ REQSPEC es una especificación de requisitos:
  - Si *dm* ∈ $DOMMOD_{N1}$, entonces *rs* consiste en un catálogo de requisitos del dominio que contiene los requisitos textuales relacionados con la definición del dominio de la línea de productos *dm*.
  - Si *dm* ∈ $DOMMOD_{N2}$, entonces *rs* contiene los requisitos de un producto concreto que se ha instanciado a partir del análisis del dominio de la línea de productos *dm*, es decir (1) los requisitos obligatorios instanciados de *dm*; (2) los requisitos alternativos y opcionales seleccionados e instanciados procedentes de *dm*; y eventualmente (3) los requisitos propios del producto, que llamaremos *deltas*, que no estaban definidos en el análisis de la línea de productos (esto es, que no pertenecen a DOMMOD).

Coloquialmente decimos que con α los modelos del marco de análisis dado por SIRENspl, que definen un espacio *n*-dimensional (donde *n* es el número de modelos utilizados), se *aplanan* en una especificación fundamentalmente textual (en una sola dimensión, por tanto) que contiene los requisitos instanciados comunes y variables de la línea de productos o los requisitos de un producto instanciado dentro de la línea de productos.

La correspondencia de aplanamiento α se debe poder usar en el marco de un modelo de procesos iterativo e incremental:

- α debe poder trabajar con modelos incompletos: no debe ser necesario instanciar todo el modelo del dominio *dm* (resolver todos los puntos de variación) para

aplicar α a $dm \in DOMMOD_{N2}$. En tal caso, *rs* también podría contener elementos no instanciados.

- Es preciso asegurar que cuando algún aspecto de la especificación de requisitos generados *rs* cambie, por ejemplo alguno de los valores de atributos de los requisitos generados se modifique, ese cambio se mantenga a pesar de que se produzca una nueva instanciación del modelo y la generación de un nuevo incremento de la especificación. Es decir, generaciones sucesivas no deberían sobreescribir los deltas en la instanciación del producto: la especificación de un producto dentro de la línea de productos se debería poder realizar de forma incremental.
- Al mismo tiempo, sería interesante disponer de *sincronización* entre *dm* y *rs*, con el objetivo de disponer de dos *vistas* para trabajar en la definición de la línea de productos o de un producto concreto, la *vista modelos* y la *vista textual*. Como hemos señalado en la Sección 4.1, estas vistas podrían jugar un papel complementario, con usos distintos para unos interesados u otros.

Esta Sección 4.3 se estructura de la siguiente manera. En primer lugar, en la Sección 4.3.2 se reflexiona sobre la naturaleza de los requisitos y su contexto, pues uno de los objetivos que nos proponemos es mejorar el contexto de la especificación de requisitos textuales. Posteriormente, para describir la propuesta de aplanamiento α, primero caracterizamos formalmente los modelos de partida que conforman DOMMOD, es decir, los modelos de análisis del dominio de SIRENspl, que acotamos a características (Sección 4.3.3.1) y casos de uso genéricos (Sección 4.3.3.2). Después analizamos los modelos destino, para lo cual caracterizamos la especificación de requisitos con SIREN (Sección 4.3.4). Estas caracterizaciones se realizan mediante la construcción de los metamodelos correspondientes con MOF. Una vez establecidos los metamodelos de origen y destino, en la Sección 4.3.5 se define formalmente la correspondencia de aplanamiento, que da lugar a (1) requisitos *directos*, que transcriben en una nueva forma la información ya especificada textualmente en los modelos de partida; y (2) requisitos que denominamos *literarios*, que en cierta forma *parafrasean* los modelos de partida en busca de hacer explícita información *codificada* (normalmente de forma gráfica) en dichos modelos. La viabilidad de la propuesta aquí formulada se muestra en la Sección 4.4, donde se recoge una implementación de la correspondencia de aplanamiento en el ámbito de MDA (*Model Driven Architecture*), que se valida a través de una aplicación retrospectiva al caso de estudio de los sistemas teleoperados para mantenimiento de cascos de buques (Capítulo 3).

4.3.2 Requisitos y contexto

La definición tradicional de requisito de IEEE, que hemos visto en la Sección 1.1.1, no dice nada sobre “el contexto” o “el entorno” que pensamos debería acompañar a los requisitos en una especificación de requisitos para hacerla más comprensible y más útil. La ausencia de este contexto es una de las críticas que se hacen a las listas de requisitos textuales: con ellas “hay poco contexto” (Sección 4.1). Para Lam *et al.* [171], “el contexto de la reutilización se debe hacer explícito para evitar su mal uso”. Con la función de aplanamiento α decíamos que buscamos una especificación de requisitos en la que “los requisitos se encuentren en contexto”. Estos razonamientos nos han llevado a reflexionar sobre qué es el contexto (o entorno) de un requisito. A este respecto son

interesantes los trabajos de Jackson sobre el significado de los requisitos funcionales, que justifican la necesidad de expresar los requisitos en contexto.

En este apartado se presentan las reflexiones de Jackson en un clásico trabajo [152] sobre la naturaleza de los requisitos y el contexto o entorno de los mismos, y se ligan tales reflexiones con nuestro enfoque de aplanamiento de los modelos de análisis del dominio de SIRENspl. Creemos que las reflexiones de Jackson sobre la naturaleza de los requisitos (1) por un lado, aportan una reflexión interesante sobre el significado de uno de los conceptos básicos usados en esta tesis doctoral, el requisito; y (2) por otro lado, justifican el interés de modelar el contexto o entorno de los mismos, presente en SIRENspl y en la especificación de requisitos SIREN generada a partir de SIRENspl. Es por esto que consideramos interesante incluir esta sección en esta memoria de tesis doctoral.

Para Jackson, el trabajo de un ingeniero de software es el desarrollo e instalación de una *máquina*: "todo o parte de un conjunto de computadores programados para comportarse en una manera que asegure la satisfacción de los requisitos". Los *requisitos*, sin embargo, no conciernen directamente a la máquina, sino al *entorno* en el que estarán instalados. El entorno es "la parte del mundo con la cual la máquina interactúa, en la que los efectos de la máquina son observados y evaluados". A partir de estas definiciones de Jackson podemos inferir que para modelar los requisitos es preciso modelar también el entorno de dichos requisitos. En SIRENspl los requisitos de la línea de productos se hallan dentro de (1) los casos de uso, fundamentalmente de los escenarios de los casos de uso; (2) las características; (3) los requisitos textuales asociados a las características; y (4) las plantillas de atributos de calidad. El contexto de estos requisitos, esto es, el entorno que les dota de auténtico significado, viene dado en SIRENspl por (1) los estímulos de los actores en los escenarios de interacción de los casos de uso; (2) las características mismas, que actúan como clusters de requisitos; (3) las características externas; (4) las características de cambio; (5) los casos de cambio; (6) los estímulos de los actores en los escenarios abstractos de las plantillas de atributos de calidad; (7) el esquema conceptual del dominio, que proporciona un vocabulario del dominio; y (8) en general, por las relaciones de traza definidas entre todos los artefactos de DOMMOD.

Según Jackson la máquina puede afectar al entorno, y también verse afectada por él, debido a que máquina y entorno comparten algunos *fenómenos compartidos (shared phenomena)*. Esto es, hay algunos eventos que suceden tanto en la máquina como en el entorno; y hay estados que son estados de ambos. Afectando directamente a fenómenos compartidos, la máquina puede (y debe) afectar indirectamente a *fenómenos privados* del entorno. Más aún, para Jackson, "los requisitos están localizados en el entorno", es decir, los requisitos son condiciones sobre los eventos y estados del entorno. Los requisitos, por tanto, pueden especificarse completamente sin referencias a la máquina.

Para Jackson, la descripción completa de un requisito consiste de al menos dos partes: (1) se debe describir el requisito mismo –la condición deseada sobre los fenómenos del entorno–; y (2) se debe describir también las propiedades *dadas* del entorno en virtud de las cuales será posible para una máquina, que participa sólo en los fenómenos compartidos, asegurar que el requisito es satisfecho. Esta distinción entre lo deseado y lo dado debe ser reflejada en dos descripciones separadas:

- Un *requisito de cliente* ρ (*customer requirement*) expresa una condición sobre los fenómenos del entorno que deseamos hacer realidad instalando la máquina.
- Un *aserto del entorno* ε (*environment assertion*) expresa una condición sobre los fenómenos del entorno que sabemos que será cierta sean cuales sean las propiedades y el comportamiento de la máquina.

En otras palabras, el requisito ρ se puede considerar *opcional*, expresando un deseo, mientras que el aserto ε es *indicativo*, expresando algo que debe ser cierto. Hay, por tanto, dos tipos de propiedades del entorno: aquellas que son garantizadas por el entorno mismo, y aquellas que son garantizadas por la máquina.

Aunque la descripción optativa ρ y la descripción indicativa ε son relevantes para los requisitos, puede parecer a primera vista que el trabajo del ingeniero de requisitos se completa cuando los requisitos del cliente ρ se han escrito. Desde un punto de vista miope esto puede parecer cierto; pero hay razones por las cuales la formulación de los asertos del entorno es también parte integral de la IR:

- En primer lugar, hacer una descripción indicativa del entorno y validarla a través de un estudio del dominio y de discusiones con expertos del dominio es una forma vital de obtener y demostrar la necesaria comprensión del entorno en el que están los requisitos de los clientes y en el cual tienen sentido.
- En segundo lugar, es necesario mostrar que el requisito se puede satisfacer por alguna máquina. Puede ocurrir que el entorno no contenga suficientes restricciones y cadenas causa-efecto para conectar apropiadamente los fenómenos compartidos con los fenómenos que son de interés para el cliente.

Para mostrar que los requisitos se pueden satisfacer por alguna máquina se lleva a cabo una *especificación* de la misma. Una especificación σ es una descripción optativa de una condición sobre los fenómenos compartidos en la interfaz entre la máquina y el entorno. Una máquina que cumpla con lo establecido por σ asegurará la satisfacción del requisito, esto es:

$$\varepsilon, \sigma \vdash \rho$$

Si la máquina cuyo comportamiento satisface σ es instalada en el entorno, y el entorno tiene las propiedades descritas en ε, entonces el entorno exhibirá las propiedades descritas en ρ.

La especificación constituye un puente entre la IR, que está relacionada con el entorno, y la ingeniería del software, que está relacionada con la máquina. La máquina puede ser tratada por el programador como un sistema formal. El entorno, en cambio, es normalmente una parte del mundo físico que no está formalizada, lo cual presenta una dificultad enorme. Estas reflexiones nos recuerdan la discusión de Goguen [119] sobre "lo seco" y "lo mojado" ("the dry and the wet"). Lo seco sería la información formalizada, *insensible al entorno* (por ejemplo, la representación de un programa en un computador), mientras que lo mojado sería la información informal, *sensible al entorno* (por ejemplo, las acciones de los interesados sobre el sistema): se trata de dos aspectos

de la información opuestos pero complementarios. La IR tiene una fuerte necesidad práctica de reconciliar lo seco con lo mojado.

Normalmente se habla del entorno usando lenguaje natural, con todas sus ambigüedades. Jackson propone el uso de la lógica para describir el entorno, buscando una distancia mínima entre el entorno y su descripción en requisitos a través de los *términos base* (*ground terms*) en los cuales basar la especificación: en esta tesis doctoral, en cambio (1) en SIRENspl usamos modelos de ingeniería del software, adoptando una visión más práctica que nos lleva a utilizar los elementos del ECD (descritos en la Sección 3.4.2) en la descripción de los casos de uso, de las características, de los requisitos textuales y de los escenarios abstractos de las plantillas de atributos de calidad: podríamos decir que en el ECD están nuestros *ground terms*; (2) en SIREN, por otro lado, proponemos el uso de lenguaje natural, de manera que nuestros *ground terms* estarían recogidos en el glosario de términos del documento de requisitos y en el modelo conceptual (si este último se hubiera realizado).

Como hemos visto, para Jackson en una especificación de requisitos es preciso diferenciar entre aspectos optativos (requisitos del cliente) e indicativos (asertos del dominio). ¿Cómo se ven reflejadas las reflexiones de Jackson en las definiciones de requisito que se utilizan en la práctica de la IR? Distintas fuentes se refieren a conceptos relacionados con estas reflexiones de maneras distintas. Sin ánimo de ser exhaustivos, a continuación ilustramos la distinción entre aspectos optativos (requisitos) y aspectos indicativos (restricciones) a través de los siguientes ejemplos:

- El estándar IEEE 1233 [139] define requisito como “una capacidad, una condición, o una restricción”.
- El estándar IEEE 830 [138] afirma que las cuestiones básicas que se han de tener en cuenta al escribir una especificación de requisitos del software son: (1) funcionalidad; (2) interfaces externas; (3) rendimiento; (4) atributos (portabilidad, corrección, facilidad de mantenimiento, seguridad, etc.); (5) restricciones de diseño (estándares, lenguaje de implementación, políticas para integridad de la base de datos, limitaciones de recursos, entornos operativos, etc.).
- Robertson y Robertson [265] distinguen en su método Volere entre requisitos funcionales, no funcionales y restricciones. Las restricciones son “cuestiones globales que perfilan los requisitos”. Son restricciones sobre el proyecto mismo o restricciones sobre el diseño. Para estos autores las restricciones no son más que otro tipo de requisito, ya que según sus palabras no se trata de una “opinión”, sino de una restricción del negocio que debe ser satisfecha por cualquier solución propuesta.
- Pfleeger [244] distingue entre requisitos funcionales, que describen una interacción entre el sistema y su entorno, y “requisitos no funcionales o restricciones”, que describen “una restricción sobre el sistema que limita nuestras elecciones para construir una solución al problema”.

Durante la revisión de SIREN realizada en el proyecto GARTIC (Sección 1.4.3.1) hemos adoptado la taxonomía de Sommerville [276], que distingue entre requisitos funcionales, requisitos no funcionales y requisitos del dominio. Pensamos que siguiendo

la terminología de Jackson, los dos primeros tipos se pueden considerar requisitos del cliente, mientras que el último tipo contiene los asertos del entorno. La taxonomía de Sommerville es sencilla y, como vemos, se relaciona de forma clara con la terminología de Jackson. Atendiendo a esta clasificación de requisitos de Sommerville, tenemos que en SIRENspl:

- Los requisitos funcionales se pueden derivar de (1) características; (2) requisitos textuales; (3) pasos realizados por el sistema en los casos de uso; y (4) mecanismos y estrategias de las plantillas de atributos de calidad.
- Los requisitos no funcionales se pueden derivar de (1) características; (2) requisitos textuales; (3) el atributo *Requisitos no funcionales* de los casos de uso; y (4) el atributo *Requisitos no funcionales* de los pasos de los casos de uso.
- Los requisitos del dominio se pueden derivar del esquema conceptual del dominio (ECD).

4.3.3 Caracterización de los modelos de análisis DOMMOD

En este apartado se describe la información que los modelos de análisis DOMMOD deben gestionar para describir adecuadamente un análisis del dominio con SIRENspl. El objetivo es que dicha caracterización sirva como base de la generación de una especificación de requisitos SIREN mediante la función de aplanamiento α descrita en la Sección 4.3.1.

DOMMOD se describe a través de dos metamodelos complementarios: (1) uno centrado en el concepto de característica (*MMD-FEAT*, Sección 4.3.3.1); y (2) otro centrado en el concepto de caso de uso genérico (*MMD-UC*, Sección 4.3.3.2). En esta tesis doctoral acotamos DOMMOD a los modelos de características y de casos de uso, que son los más relacionados con los requisitos, y dejamos fuera otros elementos de SIRENspl: el modelo conceptual, las plantillas de atributos de calidad y los comandos. El modelo de requisitos textuales destino REQSPEC, compatible con SIREN, se describe posteriormente a través de un tercer metamodelo (*MMreq*, Sección 4.3.4). Estos tres metamodelos se describen a nivel conceptual utilizando sendos diagramas de clases UML en el contexto de MOF (*Meta Object Facility*) [234]. Decimos que se describen "a nivel conceptual", de forma independiente de la plataforma, pues posteriormente será necesario trasladar estos metamodelos a la notación específica de la plataforma donde se implemente la propuesta de aplanamiento α entre DOMMOD y REQSPEC (*MediniQVT* [142] en nuestro caso, para lo cual será preciso trasladar estos metamodelos a la notación *Ecore*, ver Sección 4.4). En la Figura 38 se muestran de forma sencilla las relaciones esquemáticas de estos tres metamodelos con los dominios DOMMOD y REQSPEC descritos en la Sección 4.3.1, todavía sin incluir los detalles de la plataforma, que son propios de la implementación escogida para la propuesta de aplanamiento.

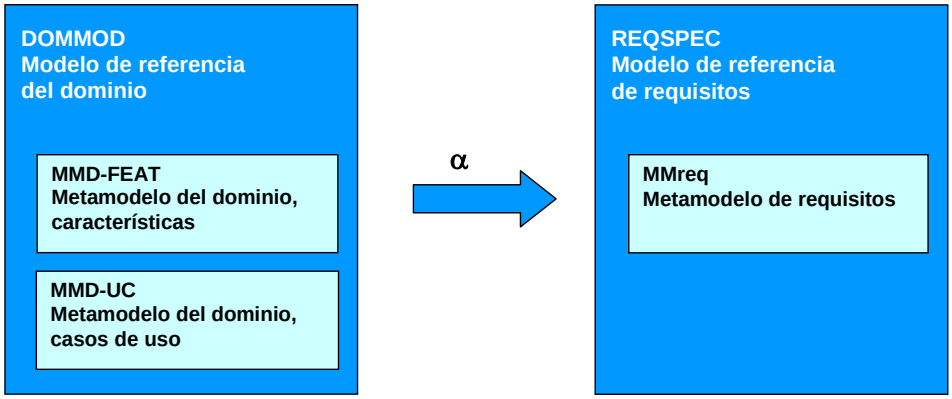

Figura 38 Descripción de los modelos de referencia de partida y destino a través de metamodelos

4.3.3.1 Metamodelo de características

En la Figura 39 se muestra el metamodelo de características de SIRENspl, a nivel conceptual, que describe la información de DOMMOD relacionada con las características. Denominamos a este metamodelo *MMD-FEAT* (*Metamodelo del Dominio-Características*). La información reflejada en este metamodelo constituye la descripción del punto de partida de las transformaciones de características hacia el modelo de referencia de requisitos textuales destino REQSPEC (Sección 4.3.4). En la Figura 39 se observa cómo las características en MMD-FEAT están contenidas en un catálogo (*Catalogue*) que está contenido a su vez en un repositorio (*Repository*): un repositorio es una composición de catálogos. A su vez, un catálogo es una composición de las características que lo definen (*Feature*) y de todos los elementos reutilizables definidos en el modelo del dominio, como los parámetros (*Parameter*), los tipos de datos (*DataType*) y los requisitos textuales (*TextualRequirement*). Como se describe a lo largo de esta sección, las características están relacionadas entre sí de diferentes formas, pueden ser de diferentes tipos y estar parametrizadas.

En la Figura 39 se observa cómo la clase central del metamodelo es *Feature (característica)*. Una característica representa un nodo de un árbol de características SIRENspl, que puede por tanto refinarse en un conjunto de 0..n características hijas, tal y como marca la relación *refine-and*. Una característica puede además estar trazada directamente a 0..n *requisitos textuales* (*TextualRequirement*) como marca la relación *trace-to*. Esta relación liga los requisitos textuales del dominio que se trazan a partir de la característica, y por tanto esta relación deberá mantenerse de alguna manera en el modelo destino MMreq. Una característica puede estar ligada por medio de relaciones de traza a cualquier otro artefacto parte de los requisitos, como las plantillas de atributos de calidad, pero en MMD-FEAT se omite su modelización por simplicidad (estas relaciones se muestran en el metamodelo de casos de uso genéricos, MMD-UC, ver Figura 40).

Una característica posee una asociación con *FeatureType* que dará lugar al atributo correspondiente *type*, que indica si se trata de una característica externa o de cambio. Una característica sólo puede pertenecer a uno de estos tipos, puesto que una característica se representa de forma excluyente en el diagrama de características: o es externa o de cambio. En función del valor de *type*, en los requisitos textuales de MMreq

se indica con un atributo la procedencia del mismo (ver atributo *dataSource* de MMreq, Sección 4.3.4).

Otra forma de distinguir entre características es por medio de *level*, su nivel de obligatoriedad (asociación con *MandatoryLevel*), distinguiendo entre características obligatorias (*MandatoryFeature*) y características opcionales (*OptionalFeature*). El valor de *level* sirve posteriormente para calcular la prioridad de los requisitos textuales generados en el destino MMreq.

La representación de los puntos de variación (*Vp-Feature*) viene marcada en MMD-FEAT por una especialización de las características, que permite a una característica formar parte de un punto de variación y además tener asociado un tipo (por medio de los atributos *level* y *type* anteriores). Una característica que represente un punto de variación en el sistema contiene una colección de *VariantFeature* (*características variantes*) que representan las alternativas entre las cuales la vp-feature correspondiente deberá instanciarse cuando se instancie un producto concreto dentro de la línea de productos. El número de variantes a instanciar por cada vp-feature está descrito por los números mínimo y máximo indicados en su referencia a cardinalidad (*AssociationCardinality*). Se permite definir las cardinalidades habituales en UML, como por ejemplo 0..1, 1..1, 1..n. y cualquier otra combinación de min..max. Las características que representen variaciones pueden ser las predeterminadas a instanciarse al resolver un punto de variación si son de la clase *DefaultVariantFeature*.

Una característica puede ser de *capacidad*, indicando una capacidad o funcionalidad que un producto instancia de la línea de productos puede ofrecer (*CapabilityFeature*), o bien puede ser de *implementación*, indicando la realización de una capacidad determinada (*ImplementationFeature*). Las características de implementación se trazan a las características de capacidad por medio de la asociación *implemented-by*.

Una característica puede ser de tipo parametrizado: se trata de una característica que queda especificada por uno o varios parámetros que se pueden instanciar a un conjunto de valores definidos de determinado *DataType*. Para representar esta posibilidad en el modelo añadimos dos nuevos subtipos de características (*ParametrizedFeature* e *InstantiatedParametrizedFeature*). Con *ParametrizedFeature* modelamos una parte de la línea de productos sin instanciar (modelado del dominio). Por el contrario con *InstantiatedParametrizedFeature* modelamos un producto concreto dentro de la línea de productos. En este contexto hablamos de parámetros (*Parameter*) y parámetros instanciados (*InstantiatedParameter*), siendo estos últimos una asociación entre un parámetro y un valor instanciado (*DataValue*). Para los parámetros distinguimos dos subtipos, *parámetros globales* y *parámetros locales*, en función del ámbito o alcance de su identificador. Los caracteres *$* y @ se usan al comienzo del nombre del parámetro para marcar su tipo, de forma que puedan ser referenciados en la descripción de la característica (@ para los locales y *$* para los globales).

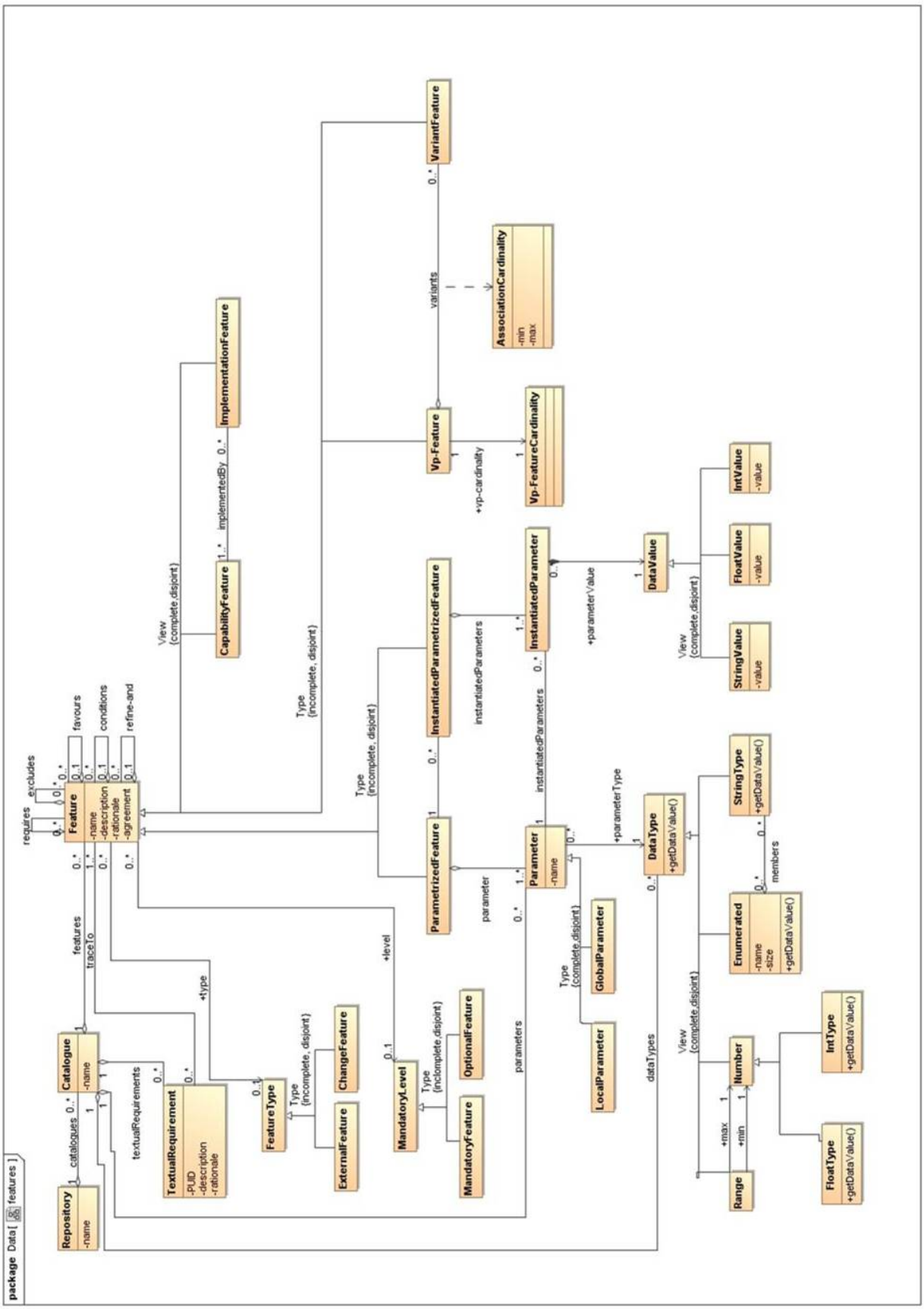

Figura 39 MMD-FEAT: Metamodelo de características (conceptual)

Los parámetros están definidos por un tipo de datos *DataType*. Los parámetros mantienen la referencia al tipo, mientras que los parámetros instanciados mantienen la referencia al valor instanciado (*DataValue*). Los tipos que se permiten son los siguientes:

- *Number*: representa un entero o un real.
- *Range*: representa un intervalo de valores o rango, mediante dos referencias a un número.
- *StringType*: representa un tipo cadena de caracteres.
- *Enumerated*: representa una colección de cadenas con nombre definido.

Los campos de la plantilla que describe la característica se especifican como atributos de la clase *Feature*. Podremos utilizar esta información para rellenar la información del requisito textual destino asociado a cada característica. Los campos son los siguientes:

- *name*: indica el nombre de la característica.
- *description*: indica la descripción textual, el significado de la característica.
- *rationale*: indica la razón por la que la característica está incluida en el proyecto.
- *agreement*: indica el nivel de acuerdo o de compromiso acordado con esa característica.

Entre los campos propuestos por la plantilla de características se encuentran las relaciones denominadas *CompositionRules* (*reglas de composición*), que definen formas adicionales que una característica tiene de relacionarse con otras. Las características se relacionan por medio de las asociaciones *favours, conditions, requires, excludes* y como hemos comentado anteriormente *refine-and.* Adicionalmente, si la característica es de tipo *Vp-Feature* entonces tiene una relación extra con sus variantes por medio de *variants.* A continuación describimos con más detalle las relaciones entre características:

- *conditions*: la característica condiciona o afecta negativamente la inclusión de un conjunto de características.
- *favours*: la característica favorece o determina positivamente la inclusión de un conjunto de características.
- *requires:* la adición de la característica requiere o necesita la inclusión de un conjunto de características.
- *excludes:* la adición de la característica excluye o imposibilita la inclusión de un conjunto de características.
- *refine-and*: la inclusión de la característica determina la inclusión de sus hijas si estas son obligatorias, siendo este un conjunto de características que depende de ella.
- *variants*: la inclusión de esta característica determina la inclusión de un subconjunto del conjunto de posibles hijos, siendo la cardinalidad de este subconjunto restringida por las cardinalidades mínima y máxima de la asociación.

Para facilitar la transformación a MMreq, metamodelo descrito en la Sección 4.3.4, se han identificado clases comunes en ambos metamodelos, a saber: *TextualRequirement*, *Parameter, Catalogue, Repository* y *DataType*. Estas clases comunes facilitan el encuadre a la hora realizar el proceso de transformación y ciertamente se encuentran en ambos dominios, por lo que nos hemos limitado a transcribir la realidad de forma que los metamodelos sean más compatibles. La clase *Textual Requirement* será descrita en mayor detalle en el metamodelo destino MMreq.

4.3.3.2 Metamodelo de casos de uso genéricos

En la Figura 40 se muestra el metamodelo de casos de uso de SIRENspl, a nivel conceptual, que describe la información de DOMMOD relacionada con los casos de uso genéricos. Denominamos a este metamodelo *MMD-UC* (*Metamodelo del Dominio-Caso de uso*). La información reflejada en este metamodelo constituye la descripción del punto de partida de las transformaciones de casos de uso hacia el modelo destino REQSPEC (Sección 4.3.4). En la Figura 40 se observa cómo los casos de uso, al igual que las características en MMD-FEAT, están contenidos en un catálogo (*Catalogue*) que está contenido a su vez en un repositorio (*Repository*): un repositorio es una composición de catálogos. A su vez, un catálogo es una composición de los elementos reutilizables del dominio, y por tanto consiste de una composición de casos de uso genéricos (*UseCase*), requisitos textuales (*TextualRequirement*), plantillas de atributos de calidad (*QualityAttributeTemplate*), comandos (*Command*) y por supuesto características (*Feature*).

La principal clase conceptual del metamodelo de casos de uso es obviamente *UseCase*. *UseCase* tiene como atributos los campos de la plantilla de especificación de casos de uso propuesta en SIRENspl. Un caso de uso puede ser de tres tipos distintos: *AlternativeUseCase*, *MandatoryUseCase* u *OptionalUseCase*. *UseCase* tiene asimismo una composición de pasos de caso de uso (*UCStep*), y puede tener además una o varias características definidas mediante la relación *nonFunctionalRequirements*. Es posible que un caso de uso contenga referencias a un conjunto de plantillas de atributos de calidad (*QualityAttributeTemplate*) y a un conjunto de comandos (*Command*). Un caso de uso está ligado a un actor primario (*Actor*), que a su vez puede generalizar varios actores (asociación reflexiva *generalization*).

La clase *Feature* y la clase *TextualRequirement* permiten ligar el metamodelo de casos de uso (MMD-UC) con el metamodelo de características (MMD-FEAT). Los casos de uso y las características están trazados mediante una asociación N:M *traceTo*, en la cual un caso de uso puede estar relacionado con una o varias características y una característica puede estar relacionada con cero o varios casos de uso. La traza con la clase *Feature* está establecida textualmente mediante la relación *nonFunctionalRequirements*. Además, es posible que uno o varios pasos del caso de uso estén trazados a características, en este caso esta relación está marcada por la conexión *traceTo* desde el paso del caso de uso a la característica. En caso de que no se quiera especificar el detalle de qué pasos del caso de uso están relacionados con qué características se utiliza la relación anterior *nonFunctionalRequirements* entre un caso de uso y una o varias características. Los *ChangeCase*, que capturan las posibles extensiones futuras del sistema, están relacionados con *ChangeFeature*.

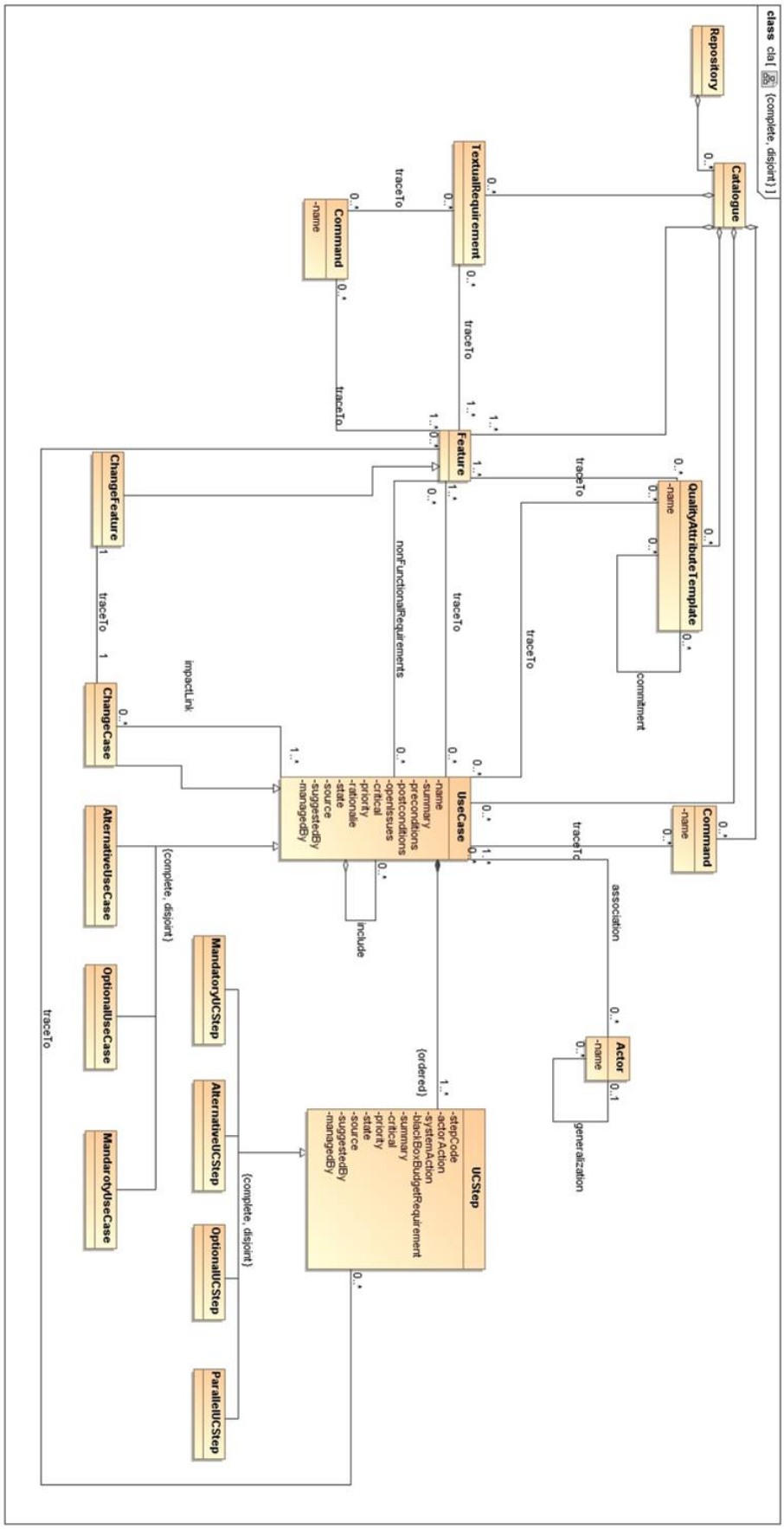

Figura 40 MMD-UC: Metamodelo de casos de uso (conceptual)

Un paso de un caso de uso se declara mediante la clase conceptual *UCStep*. Esta clase tiene como atributos: (1) *stepCode*, el número de paso, que indica el orden en que se debe ejecutar el paso en el caso de uso; (2) *actorAction*, que indica la acción que debe llevar a cabo el actor en este paso; (3) *systemAction*, que hace referencia a la acción del sistema en respuesta al estímulo de *actorAction*; (4) *blackBoxBudgetRequirement*, que se corresponde con los requisitos no funcionales relacionados con este paso; (5) *summary*, un campo que contiene la descripción del paso; y finalmente (6) los atributos de requisitos del conjunto mínimo de SIREN 2.0 [226].

El paso de un caso de uso puede ser de cuatro tipos: *obligatorio, alternativo, opcional* y *paralelo*. Un paso *obligatorio* se debe ejecutar siempre. Un paso *alternativo* especifica que se puede ejecutar una de sus variaciones. Estos pasos alternativos tienen el mismo número de paso, y se ejecutará una u otra alternativa dependiendo de la variante escogida en una vp-feature. Un paso *opcional* del caso de uso describe un comportamiento opcional del sistema. Y por último un paso *paralelo* del caso de uso se ejecuta en composición paralela, concurrentemente con otros pasos paralelos.

Por simplicidad se ha decidido acotar el dominio de manera que las plantillas de atributos de calidad y los comandos quedan especificados como “cajas negras”, pero hemos creído conveniente mostrar sus relaciones con los casos de uso y las características. Una plantilla de atributos de calidad (*QualityAttributeTemplate*) contiene la referencia a cero o varias características. Estas plantillas pueden tener relaciones de dependencia con otras plantillas de atributos de calidad, que se especifican mediante la relación *commitment*. Un caso de uso puede estar relacionado con varias plantillas de atributos de calidad.

4.3.4 Caracterización de la especificación de requisitos REQSPEC

En la Figura 41 se muestra el metamodelo de requisitos y de documentos de requisitos, a nivel conceptual, que describe la información de REQSPEC. Denominamos a este metamodelo *MMreq* (*Metamodelo de requisitos*). La información reflejada en este metamodelo constituye la descripción del punto de destino de las transformaciones desde el modelo origen DOMMOD (Sección 4.3.3). MMreq describe el modelo de referencia de requisitos textuales de SIREN, si bien se han realizado pequeñas modificaciones sobre la propuesta de SIREN 2.0 [226]. Estas modificaciones son las siguientes:

- Se ha acotado el modelo de referencia de SIREN en relación al personal implicado en el proyecto y a los tipos de catálogos, pues estas cuestiones no son relevantes en las transformaciones y hemos preferido centrarnos en los requisitos y sus relaciones.
- El objetivo de este trabajo es obtener un documento de requisitos del producto a partir de un modelo del dominio de la línea de productos, es decir, obtener un documento de requisitos instanciado (Nivel 2 de la línea de productos). Con vistas a posibles mejoras se ha conservado el catálogo para permitir extraer documentos de requisitos sin instanciar (Nivel 1 de la línea de productos), de forma que cuando el cliente desee instanciar un nuevo producto dentro de la línea de productos pueda elegir qué requisitos prefiere sobre un documento de requisitos y no sobre el modelo de características, como una especie de vista

alternativa. Se añade por tanto una nueva forma de organizar los requisitos: inicialmente sólo era posible mediante un catálogo [226], ahora también se puede acceder a los requisitos en función del documento al cual pertenecen y su sección. Por tanto, tenemos documentos organizados en secciones, las cuales pueden estar anidadas.

- Los tipos de datos a los que puede hacer referencia han sido mejorados. En la propuesta original de SIREN 2.0 [226] sólo se permite hacer referencia a literales, mientras que en MMreq hay mayor riqueza de tipos: *Range, FloatType, IntType , Enumerated* y *StringType.*
- Los parámetros han sido extendidos de forma que se permiten nuevos subtipos como locales y globales y parámetros instanciados. Aunque MMreq se ha desarrollado independientemente, la forma de distinguir entre requisitos parametrizados y parametrizados instanciados se ha validado mediante el metamodelo REMM (*REquirements-MetaModel*) de Moros *et al.* [219]. Como ya hemos comentado, estamos interesados en que el cliente pueda instanciar la variabilidad tanto en los modelos de análisis del dominio (características y casos de uso) como en el documento de requisitos, de la manera que le resulte más cómoda. Para ello se distingue entre requisitos parametrizados y requisitos parametrizados instanciados (*ParametrizedRequirement* e *InstantiatedParametrizedRequirement*, Figura 41), y entre parámetros y parámetros instanciados (*Parameter* e *InstantiatedParameter*, respectivamente, ver Figura 41).

El diagrama de la Figura 41 está centrado en la clase *requisito textual* (*TextualRequirement*). SIREN obliga a la definición de 13 atributos que conforman el conjunto mínimo de atributos de un requisito (ver Sección 2.6.2) y permite extender este conjunto mínimo. Por compatibilidad con SIREN, los requisitos no tienen nombre en MMreq.

Las relaciones de traza (*Trace*, ver Figura 41) permiten modelar diferentes relaciones entre requisitos [226]:

- *exclusive (exclusiva)*: un requisito está en conflicto con un conjunto de requisitos, y recíprocamente, de manera que sólo puede ser instanciado un requisito en todo el conjunto. En este punto extendemos la definición original de SIREN [226], en la cual la relación *exclusive* se especifica forzosamente sólo entre dos requisitos, dos a dos. En el proceso de aplanamiento entre DOMMOD y REQSPEC se generan requisitos exclusivos cuando se traslada a requisitos un punto de variación (*Vp-Feature*), resultando ser todas sus *VariantFeature* exclusivas entre sí.
- *parent/child* (*padre/hijo)*: se usa para describir un requisito más general por medio de una serie de requisitos más específicos. En ocasiones puede utilizarse para describir características alternativas, donde los refinamientos alternativos serían hijos del mismo padre, en el que textualmente se indica que los hijos son alternativos y de qué manera.

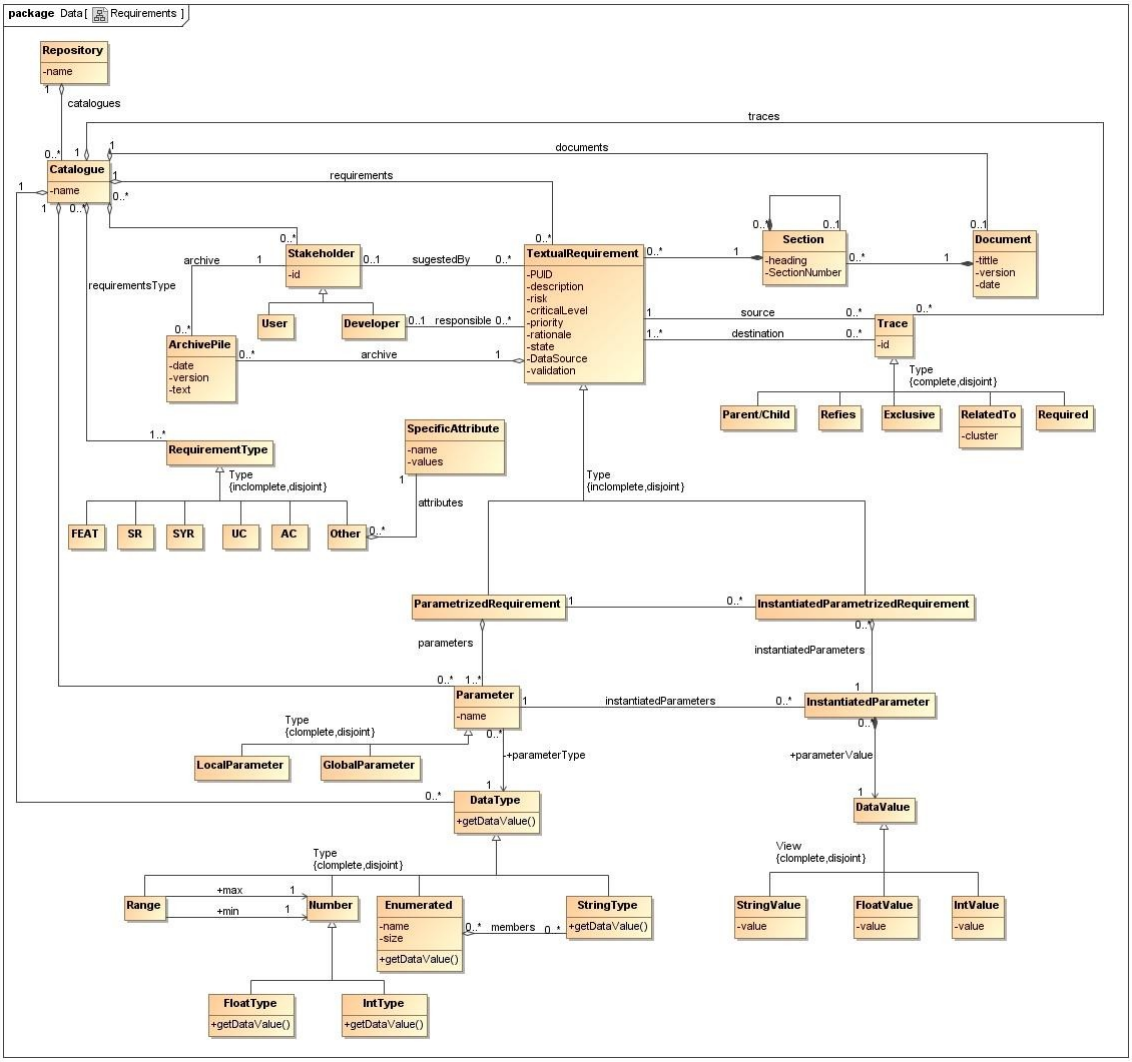

Figura 41 MMreq: Metamodelo de requisitos textuales (conceptual)

- *required (requerida)*: es una relación de dependencia direccional entre requisitos: uno de los requisitos es precondición para el otro. En el proceso de aplanamiento podría generarse una relación de este tipo cuando una característica se refina en subcaracterísticas. Las subcaracterísticas dependen de la existencia de la característica padre, luego cada una de ellas tendría una relación *required* con su característica padre (sin embargo para no ser redundante con la traza *ParentChild* no se generan).

- *relatedTo (relacionadaCon)*: indica que un conjunto de requisitos están muy relacionados entre sí. En SIREN permite definir los llamados *clúster de requisitos*. Cuando se reutiliza un requisito de un clúster, se presentan todos los requisitos relacionados con el primero, de manera que se pueden reutilizar o no. (En el caso de la relación *required*, la reutilización de tales requisitos sería obligatoria.) Estos clusters pueden habilitar "vistas de requisitos" que definen una forma de organizar los requisitos alternativa a la del documento. En el proceso de aplanamiento se podría considerar que los resultados de la traducción a requisitos textuales de las características de tipo capacidad o implementación formarán parte del mismo clúster, capacidad o implementación, respectivamente.

- *reifies (realiza)*: relación entre un requisito y un "artefacto" posterior del desarrollo, que puede ser una clase, un módulo, un componente, etc.

John [154] propone un completo metamodelo de documentos de requisitos, pero por simplicidad en MMreq se propone una forma sencilla de realizar la organización de un documento abstracto de requisitos a través de secciones de documento, de forma que una sección está compuesta de una colección de requisitos relacionados y de una colección de secciones, y un documento está formado por una colección de secciones (ver Figura 41). Así pues con las secciones se dispone de otra forma de mantener los requisitos relacionados, además de con los clusters.

El catálogo (*Catalogue*) es el elemento aglutinador de todos los elementos reutilizables de MMreq. El catálogo permite compartir la metainformación definida (tipos de datos, requisitos reutilizables, parámetros, tipos de requisitos, interesados y trazas, ver Figura 41), de manera que esta información esté disponible para ser reutilizada en otros proyectos. Gracias a mantener el catálogo con requisitos y metainformación se puede representar el documento de requisitos con estructuras alternativas a la de las secciones. Por ejemplo, podemos recorrer los requisitos a través de las relaciones de traza o por medio de los requisitos asociados a un interesado concreto.

4.3.5 Relación de aplanamiento

Una vez que hemos estudiado con todo detalle los dominios de inicio y destino DOMMOD y REQSPEC, en este apartado describimos la correspondencia de aplanamiento propuesta. En este apartado nos centramos en la esencia de la transformación, sin especificar detalles que consideramos menos importantes, como los valores de ciertos atributos de los requisitos textuales generados (al presentar las reglas de transformación QVT-*relations* en la Sección 4.4 se muestran todos los detalles de las transformaciones). Como ya hemos comentado, DOMMOD se acota para incluir (1) el modelo de características (incluyendo los requisitos textuales asociados a las características en DOMMOD); y (2) el modelo de casos de uso genéricos.

En esta sección se tratan de forma diferenciada los dos tipos de requisitos textuales generados mediante la función de aplanamiento α: (1) requisitos "directos", en el sentido de que *transcriben* sin cambios cuestiones que ya están especificadas en lenguaje natural en el modelado (Sección 4.3.5.1); y (2) requisitos "literarios", que consideramos que *parafrasean* los modelos de ingeniería del software pues expresan en lenguaje natural ciertas cuestiones ya mostradas diagramáticamente en los modelos de ingeniería del software o que se pueden inferir a partir de la representación diagramática de los modelos (Sección 4.3.3.2).

4.3.5.1 Generación de requisitos directos

En este apartado caracterizamos una especificación de requisitos $rs \in$ REQSPEC, generada mediante α a partir de un análisis del dominio instanciado $dm \in \text{DOMMOD}_{N2}$: *rs* se define como una 4-tupla (*FEAT, REQ, UC, DOC*), donde *FEAT* es el conjunto de requisitos textuales generados a partir de las características de DOMMOD; *REQ* es el conjunto de requisitos textuales ya especificados en DOMMOD (los trazados desde las características según MMD-FEAT); *UC* es el conjunto de requisitos textuales generados a partir de los casos de uso de DOMMOD; y *DOC* es el

conjunto de todos los documentos de requisitos abstractos con una estructura sencilla formada por secciones y subsecciones.

Dada una característica $f \in dm$, si α (*f*) = *r* entonces tenemos que *f* se corresponde con un requisito textual $r \in FEAT$, donde *r* es una 4-tupla (*type*, *puid*, *description*, *atrset*), en la que (1) *type* = 'FEAT'; (2) *puid* es el identificador único del requisito *r*, es decir *puid* = *n* y $\neg\exists$ *r2* $\in$ *FEAT* | (puid (r2)= n) $\wedge$ (r $\neq$ r2); (3) *description* es el texto del requisito *r* de manera que *description*(*f*) = *description* (*r*); (4) *atrset* es el conjunto de atributos de *r*, definido como *atrset* = *MIN* $\cup$ *FEAT_ATR*, donde *MIN* es el conjunto mínimo de atributos de requisitos de SIREN (Sección 2.6.2), que proporcionan información sobre la gestión de los requisitos: prioridad, grado de consecución, etc.; y *FEAT_ATR* es el conjunto de atributos que proceden de los campos de la plantilla de descripción de características, exceptuada *description*. Sea *at* $\in$ atrset, entonces *at* es una dupla (*atr_def*, *value*), donde *atr_def* es el nombre del atributo y *value* el valor asignado en *r* al atributo correspondiente.

Dada una característica de cambio $chf \in dm$, si α (*chf*) = *r* entonces tenemos que *chf* se corresponde con un requisito textual $r \in FEAT$, donde *r* es una 4-tupla (*type, puid, description, atrset*), en la que *type* = 'FUTURE_REQUIREMENT' y el resto de los elementos de la tupla son iguales que para cualquier característica. Como vemos las características de cambio de *dm* se etiquetan al aplanar con el tipo FUTURE_REQUIREMENT en lugar de FEAT ya que no son propiamente requisitos, sino extensiones anticipadas del sistema.

Dado un requisito $texreq \in dm$, si α (*texreq*) = *r* entonces tenemos que *texreq* se corresponde con un requisito textual $r \in REQ$, donde *r* es una 4-tupla (*type, puid, description, atrset*), en la que (1) *type* = 'TEXREQ'; (2) *puid* es el identificador único del requisito *r*, es decir *puid* = *n* y $\neg\exists$ *r2* $\in$ *REQ* | (puid (r2)= n) $\wedge$ (r $\neq$ r2); (3) *description* es el texto del requisito *r* de manera que *description*(*texreq*) = *description* (*r*); (4) *atrset* es el conjunto de atributos de *r*, definido como *atrset* = *MIN*, donde *MIN* es el conjunto mínimo de atributos de requisitos de SIREN. Sea *at* $\in$ atrset, entonces *at* es una dupla (*atr_def*, *value*), donde *atr_def* es el nombre del atributo y *value* el valor asignado en *r* al atributo correspondiente.

Dado un caso de uso $uc \in dm$, si α (*uc*) = *r* entonces tenemos que *uc* se corresponde con un requisito textual $r \in UC$, donde *r* es una 5-tupla (*type*, *puid*, *description*, *reqsteps*, *atrset*), en la que (1) *type* = 'UC'; (2) *puid* es el identificador único del requisito *r*, es decir *puid* = *n* y $\neg\exists$ *r2* $\in$ *UC* | (puid (r2)= n) $\wedge$ (r $\neq$ r2); (3) *description* es el texto del requisito *r* de manera que *summary*(*uc*) = *description* (*r*); (4) *reqsteps* es la secuencia de requisitos *uc_step_req* que se derivan de los pasos *ucstep* de *uc*, de manera que *uc_step_req* es la 4-tupla *(name(uc)+'.'+stepnumber(ucstep), '(' + actorAction (ucstep) + ')', systemAction (ucstep), atr_ucstep_set),* donde *atr_ucstep_set* es una terna formada por (MIN, *blackBoxBudgetRequirement(ucstep), traceTo(ucstep))*, y donde *MIN* es el conjunto mínimo de atributos de SIREN*;* (5) *atrset* es el conjunto de atributos de *r*, definido como *atrset* = *MIN* $\cup$ *UC_ATR*, donde *MIN* es el conjunto mínimo de atributos de requisitos de SIREN, que proporcionan información sobre la gestión de los requisitos: prioridad, grado de consecución, etc.; y *UC_ATR* es el conjunto formado por los campos de la plantilla de casos de uso, exceptuado *summary*. Sea *at* $\in$ atrset,

entonces *at* es una dupla (*atr_def*, *value*), donde *atr_def* es el nombre del atributo y *value* el valor asignado en *r* al atributo correspondiente.

Un documento abstracto *doc* ∈ *DOC* es una 4-tupla (*title, version, date, secseq*), donde (1) *title* es el nombre del documento; (2) *version* el número de versión; (3) *date* la fecha de última modificación del documento; y (4) *secseq* es una secuencia de secciones *sec*, donde *sec* se define como una 4-tupla (*heading, sectionNumber, subseq, reqs*), donde (4.1) *heading* es el nombre de la sección; (4.2) *sectionNumber* el número de la sección; (4.3) *subseq* es una secuencia de números de subsecciones contenidas en *sec*; y (4.4) *reqs* es un conjunto de requisitos contenidos en *sec*.

4.3.5.2 Generación de requisitos literarios

En este apartado se define un conjunto inicial de patrones sencillos con la intención de llevar a la práctica un enfoque en línea con ciertas propuestas descritas en profundidad en la revisión sistemática mostrada en la Sección 4.2 y que hemos considerado especialmente interesantes: (1) la propuesta de Arlow y Neustadt [30] sobre modelado literario; (2) la de Maiden *et al.* [196] sobre generación de requisitos textuales a partir de diagramas SD de i*; y (3) la de Meziane *et al.* [215] sobre diagramas de clases UML. En esta tesis doctoral pretendemos trasladar estos trabajos al ámbito de los diagramas de características y de casos de uso genéricos de SIRENspl, generando requisitos textuales "literarios" que "parafrasean" los diagramas, expresando con texto en lenguaje natural ciertas cuestiones mostradas diagramáticamente que podrían pasar desapercibidas a algunos interesados. Este tipo de requisitos, por ejemplo, podrían ayudar a clarificar el sistema ante la firma de un contrato. Recordemos que uno de los objetivos de Meziane *et al.* [215] es ofrecer a los usuarios en cualquier momento dos vistas diferentes de la especificación del sistema: UML y lenguaje natural. En el contexto del ejemplo que se presenta, sobre un sistema universitario, Meziane *et al.* proponen la generación de requisitos textuales como los siguientes: "*una persona es un profesor o un estudiante*", "*un profesor tiene un nombre, un domicilio, estacionamiento, fecha de nacimiento, un seminario y un seminario de supervisión*", "*cero o muchos profesores imparten cero o muchos seminarios*", "*una persona vive en una dirección*", "*una dirección tiene una calle, ciudad, estado y código postal*" y así sucesivamente. Las conclusiones al revisar el trabajo de Maiden *et al.* [196] son similares, si bien la notación empleada, diagramas SD en el contexto de i*, es menos conocida –con carácter general– que UML, y por ello en este párrafo nos hemos limitado a ejemplificar el trabajo de Meziane *et al.*

Para el modelo de características, la generación de requisitos textuales literarios que proponemos está basada en los siguientes patrones de requisitos textuales:

> P1. En el caso de que una característica *feature* no tenga padre simplemente la traduciremos al requisito: "*feature.description*".

El anterior se podría considerar como un requisito directo; se incluye por completitud.

Es posible que una característica sea externa al sistema, opcional y/o un futuro punto de extensión del sistema. Estas particularidades pueden darse simultáneamente.

> P2. Es posible que esta característica que no tiene padre sea externa, en este caso se utilizará el patrón: "*feature.description* is external to the system"

P3. Si esta característica es opcional: "*feature.description* is optional"

P4. Si esta característica es un posible futuro punto de variación: "*feature.description* is a possible variation point of the system"

P5. Si esta característica es externa y a su vez un posible futuro punto de variación del sistema, tenemos: "*feature.description* is external and a possible variation point of the system"

P6. Si es externa y opcional: "*feature.description* is external and optional"

P7. Si es externa, opcional y un posible futuro punto de variación del sistema: "*feature.description* is external, optional and a possible variation point of the system"

P8. Si es opcional y un posible futuro punto de variación del sistema: "*feature.description* is optional and a possible variation point of the system"

P9. En el caso de que una característica padre (*feature_parent*) implique a otra (*feature_child*) aplicaremos el patrón: "The feature *feature_parent.description* necessarily implies *feature_child.description"*

P10. Si esta característica hija es externa al sistema se aplica el patrón: "The feature *feature_parent.description* necessarily implies *feature_child.description* which is external to the system"

P11. Si es opcional: "The feature *feature_parent.description* necessarily implies *feature_child.description* which is optional"

P12. Si es un posible futuro punto de variación del sistema: "The feature *feature_parent.description* necessarily implies *feature_child.description* which is a possible variation point of the system*"*

P13. Si es externa al sistema y además un posible futuro punto de variación del sistema: "The feature *feature_parent.description* necessarily implies *feature_child.description* which is external and a possible variation point of the system"

P14. Si es externa y opcional: "The feature *feature_parent.description* necessarily implies *feature_child.description* which is external and optional"

P15. Si es externa, opcional y un posible futuro punto de variación del sistema: "The feature *feature_parent.description* necessarily implies *feature_child.description* which is external, optional and a possible variation point of the system*"*

P16. Si es opcional y un posible futuro punto de variación del sistema: "The feature *feature_parent.description* necessarily implies *feature_child.description* which is optional and a possible variation point of the system"

P17. Por otro lado hay características (*feature*) que están trazadas directamente a requisitos textuales (*req*). Cuando se produzca este caso se denotará de la siguiente forma: "The requirement *req.description* is traced from the feature *feature.description*"

Para el modelo de casos de uso, los patrones que se proponen son los siguientes:

P18. Un caso de uso obligatorio *UseCase* con una descripción *description* ejecutado por un actor *actor* generará el siguiente requisito textual: "The actor *actor.name* starts the use case *UseCase.name* which is summarized in: *UseCase.summary*"

P19. Un caso de uso que sea alternativo se traducirá por: "The actor *actor.name* starts the use case *UseCase.name* which is alternative and is summarized in: *UseCase.summary*"

P20. Un caso de uso opcional: "The actor *actor.name* starts the use case *UseCase.name* which is optional and is summarized in: *UseCase.summary"*

P21. Es posible que un caso de uso sea un posible futuro punto de extensión del sistema (*ChangeCase*), en este caso el requisito textual generado será: "The actor *actor.name* starts the use case *ChangeCase.name* which is a possible future extension of the system and is summarized in: *ChangeCase.summary*"

P22. Un caso de uso que es un posible punto de extensión del sistema (*ChangeCase*) puede causar un impacto en otro caso de uso *UseCase*, en este caso se generará el siguiente requisito: "Future adoption of *ChangeCase.name* may cause changes in *UseCase.name*"

Los pasos *UCStep* de los casos de uso *UseCase* con un número de paso *UCStep* y una descripción *description* pueden ser obligatorios, alternativos, opcionales o paralelos:

P23. Si el paso del caso de uso es obligatorio: "(*UseCase.name*) *UCStep.stepCode*.(*UCStep.actorAction*) *UCStep.summary*"

P24. Si es alternativo: "(UseCase.name) UCStep.stepCode. (UCStep.actorAction) Alternative.*UCStep.summary*"

P25. Si es opcional: "(UseCase.name)UCStep.stepCode.(UCStep.actorAction) Optional.*UCStep.summary*"

P26. Y si es paralelo: "(UseCase.name) UCStep.stepCode. (UCStep.actorAction) Parallel.*UCStep.summary*"

P27. Es posible que un actor *actor* sea generalizado por otro actor padre *actor_parent*, en este caso el patrón seguido es: "*actor.name* is generalized *by actor_parent.name*"

La anterior es una primera propuesta de patrones literarios para los diagramas de características y de casos de uso genéricos de SIRENspl. Es evidente que sería posible identificar más patrones, especialmente explotando la metainformación (atributos) asociada a características y casos de uso, pero consideramos esta primera lista de patrones suficiente para ilustrar el enfoque que se propone.

4.4 Implementación y validación de la propuesta de aplanamiento

En esta sección se muestra una implementación inicial de la correspondencia de aplanamiento de los modelos de análisis de SIRENspl, acotada como ya hemos dicho a los modelos de características y de casos de uso genéricos. Esta implementación se ha realizado a través de los proyectos fin de carrera de Jorge Hernández [135] y Ana Pla [248]. En estos dos trabajos la generación de requisitos textuales se desarrolla por medio de técnicas de transformación de modelos en el ámbito de la ingeniería del software dirigida por modelos (MDE, *Model Driven Engineering*) y más concretamente de MDA (*Model Driven Architecture*) [233] (Sección 4.4.1).

El desarrollo de la implementación ha sido el siguiente: (1) en primer lugar se selecciona el lenguaje de transformación *QVT* (*Query/View/Transformation*) [235] junto con una plataforma de soporte, *EMF (Eclipse Modelling Framework)* y *mediniQVT* (secciones 4.4.2 y 4.4.3); (2) acto seguido los metamodelos MOF del dominio de partida MMD y destino MMreq, a nivel conceptual, se trasladan a la plataforma de desarrollo seleccionada a través del lenguaje *Ecore* (secciones 4.4.4, 4.4.5 y 4.4.6); (3) a partir de estos metamodelos de inicio y destino refinados se define un conjunto de transformaciones QVT-*relations* que proporcionan un soporte inicial del proceso de aplanamiento (Sección 4.4.7); (4) para verificar y validar la implementación de las transformaciones de aplanamiento, el desarrollo realizado se ha aplicado retrospectivamente en los modelos resultado del caso de estudio de los sistemas teleoperados para mantenimiento de cascos de buques (STO) (Sección 4.4.8); y (5) con vistas a la aplicación de esta propuesta en el marco de un modelo de procesos iterativo e incremental, se propone un mecanismo de sincronización de los modelos de inicio y destino, tras un breve repaso del estado del arte y un estudio sobre la trazabilidad de los artefactos involucrados en las transformaciones. Este mecanismo de sincronización se vuelve a validar en el caso de estudio anterior (Sección 4.4.9).

4.4.1 MMD y MMreq en el ámbito de MDA

Como es ampliamente conocido, un metamodelo es en esencia un modelo que contiene un conjunto de reglas o especificaciones que permiten definir un modelo. En el ámbito de la ingeniería del software dirigida por modelos (MDE, *Model Driven Engineering*), una trasformación de modelos tiene por objeto especificar la consecución de un conjunto de modelos objetivo a partir de un conjunto de modelos fuente. OMG (*Object Management Group*) proporciona un estándar para definición de meta-metamodelos que se conoce como *Meta Objet Facility* (MOF 2.0) [234], que hemos aplicado en las

secciones anteriores para definir MMD y MMreq, a nivel conceptual, con la notación de los modelos de clases de UML, que es compatible con MOF. MOF propone una jerarquía de cuatro niveles de modelos atendiendo a su nivel de abstracción. El nivel más alto en la jerarquía MOF es el nivel M3 o de meta-metamodelado. En este nivel se proporcionan las especificaciones necesarias para modelar metalenguajes de forma abstracta. Tomando como base este nivel se definen metalenguajes como el metamodelo de UML, que permite definir nuestros propios metamodelos. Esto es lo que se conoce como nivel M2 o nivel de metamodelos. Debajo de este nivel se encuentran los modelos UML definidos a partir de un metamodelo en concreto. Se trata por tanto de modelos UML que describen dominios concretos y que se corresponden con el nivel M1. Por debajo del mismo se sitúan los objetos del mundo real o la información de la aplicación. Se trata de los modelos instanciados en el nivel M0. En la Figura 42 podemos ver la pirámide de la arquitectura de niveles de modelado adaptada de la documentación oficial de MOF [234].

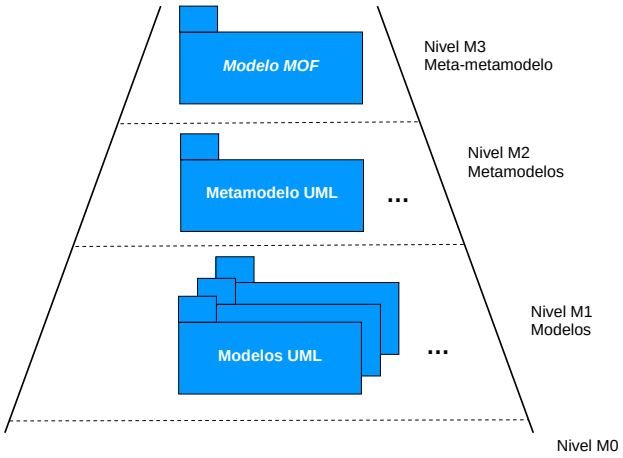

Figura 42 Arquitectura de Metadatos MOF (adaptado de [234])

Los datos en los que se aplica la correspondencia de aplanamiento se encuentran a nivel M1 o nivel de modelos, mientras que las transformaciones se definen a nivel M2 o nivel de metamodelos. En la Figura 43 observamos que las instancias conformes al metamodelo MMD, denominadas MMD_i, se corresponden a instancias $MM_{req\text{-}i}$ conforme al metamodelo MM_{req}, a través de la ejecución de la transformación MMD2MMreq.

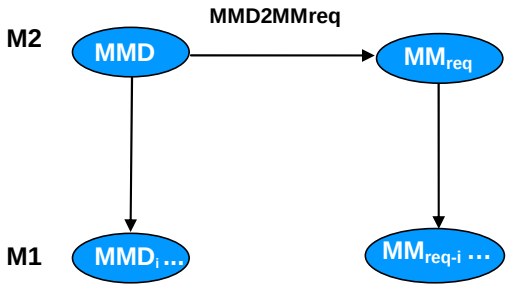

Figura 43 Correspondencia de la transformación a niveles MOF

MMreq, el metamodelo de requisitos destino que describe el modelo de referencia de requisitos de SIREN 2.0, se puede considerar compuesto por dos metamodelos relacionados entre sí, *MMtexreq* y *MMreqdoc*: (1) en MMtexreq se describen en profundidad los requisitos textuales individuales SIREN y sus relaciones con el resto de artefactos del desarrollo; y (2) en MMreqdoc los requisitos se encuadran en una estructura genérica de documento de requisitos, que posteriormente se puede trasladar a cualquier formato (IEEE 830, Volere, etc.). Las relaciones de transformación que perseguimos definir (ver Figura 44) se nombran como el modelo de origen seguido del modelo de destino, por ejemplo *MMD2MMreqdoc*[1]).

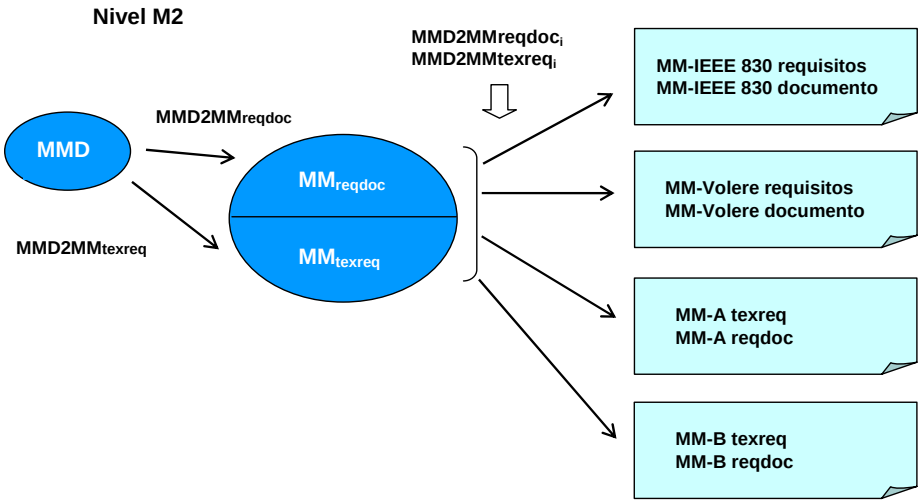

Figura 44 Relaciones de transformación a Nivel M2

En la Figura 44 se definen dos relaciones de transformación: (1) *MMD2MMreqdoc*, que tiene como partida el metamodelo del dominio para generar un documento de especificación de requisitos; y (2) *MMD2MMtexreq*, que tiene como partida el metamodelo del dominio y extrae requisitos textuales individuales. Por tanto una relación de transformación se ocupa de la estructura del documento mientras que otra es la responsable de extraer los requisitos de la entrada: ambas son complementarias, con el objetivo de obtener un documento de requisitos con contenido. En esta implementación inicial de la propuesta de aplanamiento, sin embargo, hemos decidido no considerar estructuras complejas de documentos de requisitos como la propuesta de metamodelo de documentos de requisitos de John [154]. Por tanto, en esta implementación inicial de la correspondencia de aplanamiento MMreqdoc es muy simple y se integra de forma natural en MMtexreq sin aumentar significativamente la complejidad, por lo que conviene a efectos prácticos unir ambos metamodelos en un sólo modelo destino que seguiremos denominando MMreq.

[1] La nomenclatura *MMD2MMreqdoc* simboliza metamodelo del dominio "hacia" metamodelo de documentos de requisitos.

4.4.2 Selección del entorno de desarrollo

En esta sección se presentan brevemente ATL y QVT, las dos alternativas estudiadas para la implementación de las transformaciones de aplanamiento, que probablemente son las dos herramientas más maduras que se pueden encontrar en el mercado.

4.4.2.1 ATL

ATL (*ATLAS Transformation Language*) [31] consiste de un lenguaje de transformación de modelos y de un *kit* de herramientas desarrollado por ATLAS Group –INRIA (http://www.inria.fr/) & LINA (http://www.sciences.univ-nantes.fr/lina/atl/)–. El proyecto ATL nace como alternativa a QVT. ATL proporciona la manera de obtener un conjunto de modelos objetivo a partir de un conjunto de modelos fuente, tratándose de un híbrido imperativo y declarativo, de forma que las correspondencias (*mappings*) simples pueden ser desarrolladas rápidamente de forma declarativa y si la transformación se vuelve compleja pueden ser tratadas de forma imperativa si se estima conveniente. Un programa de transformaciones ATL está compuesto de una serie de reglas que indican cómo los modelos fuente son "encajados" y navegados para crear e inicializar los objetos de los modelos destino.

ATL se ejecuta sobre una máquina virtual por lo que el entorno de ejecución mantiene un cierto nivel de flexibilidad. Como consecuencia de ello en ATL las transformaciones entre metamodelos son directamente ejecutables una vez traducidas a *bytecode*.

4.4.2.2 QVT 2.0

QVT (*Query View Tranformation*) 2.0 [235] es el estándar para la transformación de modelos propuesto por OMG, relacionado con otros estándares de OMG como MOF 2.0 y OCL 2.0.

La arquitectura QVT es de naturaleza híbrida entre el paradigma imperativo y el declarativo, siendo la parte declarativa una arquitectura de dos capas. Las capas de la parte declarativa son:

- Un metamodelo y un lenguaje fácil de usar llamado *relaciones* (*relations*), para especificar *correspondencias* (*matching*) de patrones y plantillas de creación de objetos de forma sencilla. Las trazas entre los elementos implicados en la transformación son creadas implícitamente.
- Un metamodelo y un lenguaje llamado *Core*, definido con la mínima extensión de MOF y OCL. Todas las clases de traza se definen explícitamente como modelos MOF, y la creación y borrado de instancias de traza se definen de la misma forma que el resto de los objetos implicados en la transformación.

A continuación se examinan estos elementos con más detalle:

- *Relations (relaciones):* se trata de una especificación declarativa entre modelos MOF. El lenguaje *relations* soporta correspondencias complejas de objetos y patrones, e implícitamente crea las clases de traza y sus instancias para registrar qué ocurre durante la ejecución de las transformaciones. Las relaciones pueden utilizar otras relaciones como asertos y también mantener las relaciones de

correspondencia entre los objetos encajados por cada relación. La semántica de las relaciones es una mezcla de inglés y lógica de predicados de primer orden.

- *Core (núcleo):* se trata de un modelo/lenguaje muy pequeño que sólo soporta correspondencia entre patrones con un conjunto plano de variables por medio de evaluación de condiciones sobre un conjunto de modelos. Es igual de potente que la parte *relations*, pero ha sido definido de forma más simple por lo que su sintaxis también es menos expresiva. Las trazas deben declararse explícitamente, no ocurre como en *relations* que son deducidas automáticamente a través del código de la transformación. Las *relations* pueden ser traducidas a *core* para su transformación, ya que son semánticamente equivalentes.
- *Operational mappings (correspondencias operacionales):* especificado para proveer una forma de realizar implementaciones imperativas. Contiene una extensión de OCL para permitir efectos colaterales en los valores de las variables y su sintaxis resulta más familiar a los programadores acostumbrados a programar en el paradigma imperativo.
- *Black Box Operations (operaciones de caja negra):* las operaciones MOF pueden ser llamadas desde un programa en *relations* o en *core*, haciendo posible la conexión a modo de *plugin* entre una implementación y el nombre de una operación MOF, con los siguientes beneficios: (1) permitir implementar algoritmos complejos en cualquier lenguaje; (2) permitir utilizar bibliotecas específicas de dominios concretos para calcular valores en los modelos; y (3) permitir que algunas partes de la transformación se realicen de manera opaca.

4.4.3 Marco tecnológico de desarrollo del proyecto

4.4.3.1 Marco tecnológico seleccionado

Para realizar la implementación de las transformaciones de aplanamiento entre MMD y MMreq es preciso elegir un lenguaje y una herramienta de desarrollo. En esta implementación inicial se ha escogido QVT como lenguaje de transformación por las siguientes razones:

- QVT 2.0 constituye el estándar de OMG para la transformación de modelos; ATL nace posteriormente como desarrollo privado y sujeto al proyecto Eclipse, si bien reconocemos que ha tenido un impacto importante en la comunidad de MDE.
- QVT ha madurado ha lo largo de los años y en la versión 2.0 de QVT se mejoran muchos aspectos de la misma.
- Existen herramientas de código abierto (*open source*) para Eclipse válidas para el desarrollo de proyectos en QVT. El desarrollo con este lenguaje puede realizarse en abierto e incluso colaborar en la mejora de las herramientas; además próximamente se prevé la aparición de entornos mejorados de soporte a QVT.
- Se mantiene compatibilidad con otros trabajos realizados en el seno del Grupo de investigación en Ingeniería del Software (GIS) con los que podría haber interacción en un futuro.

Las transformaciones de aplanamiento parecen intuitivamente una correspondencia sencilla entre MMD y MMreq, y por tanto se ha decidido implementarlas con un estilo declarativo, de forma que el desarrollo se pudiera desarrollar de forma ágil y con más rapidez que realizando la implementación en modo operacional (imperativo). Además la especificación gráfica de patrones en QVT-*relations* es de muy fácil comprensión.

Finalmente, la herramienta concreta elegida para el desarrollo de este trabajo ha sido *mediniQVT* [142], una herramienta de software libre que es fiel a la especificación estándar del lenguaje QVT en modo relacional.

Una transformación definida en QVT-*relations* exige la definición de dominios o metamodelos. Para MMD y para MMreq se han definido sendos ficheros *Ecore* basados en XMI [236] conforme al estándar propuesto para el intercambio de metadatos, normalmente modelos. *Ecore* es el lenguaje basado en XMI para definir modelos dentro del framework *EMF* (*Eclipse Modeling Framework*) [94][2]. EMF es un framework para Java que incorpora facilidades para la generación de código en herramientas y otras aplicaciones que estén basadas en modelos estructurados.

En la Figura 45 se muestran las principales herramientas utilizadas en la implementación de la correspondencia de aplanamiento. Los modelos MOF elaborados, MMD y MMreq, se han adaptado primero a la sintaxis definida por Ecore. Posteriormente estos modelos se han incorporado a un proyecto mediniQVT como modelos fuente y destino de las transformaciones. Después se han implementado las transformaciones en QVT-*relations* dentro de la herramienta mediniQVT. En la Figura 45 también se muestra la conexión con SirenSPLTool, un editor gráfico para SIRENspl desarrollado en GMF (*Graphical Modeling Framework*) [118] (Sección 3.5).

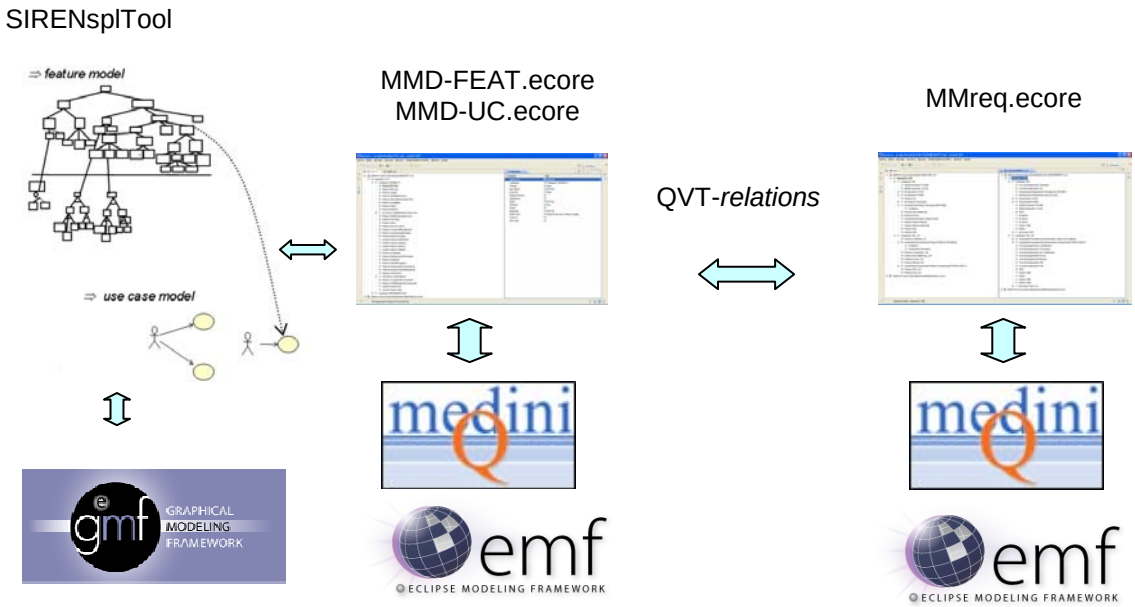

Figura 45 Marco de implementación de la correspondencia de aplanamiento

[2] *Ecore* está basado en un subconjunto de MOF 2.0, de forma que utiliza el núcleo (core) de la API del metamodelo MOF; para no confundirlo con éste se le llamó *Emf-core*, y de manera abreviada *Ecore*.

Con QVT el programa almacena internamente relaciones entre los metamodelos definidos MMD y MMreq y entre los modelos instanciados o datos. Esta última información son las llamadas *trazas de ejecución* y serán fundamentales para realizar la sincronización entre los artefactos generados.

4.4.3.2 Limitaciones de EMF

EMF facilita la tarea de llegar desde los modelos a código Java de una manera rápida, correcta, eficiente y fácilmente configurable. Sin embargo, con vistas a agilizar el proceso de codificación (del diseño a las clases), en la arquitectura de EMF se han realizado algunos recortes en la notación de MOF soportada. Por tanto, EMF se puede considerar como un subconjunto de MOF. Estas son las limitaciones principales en el diseño de los metamodelos con EMF:

- EMF no dispone de asociaciones bidireccionales, aunque sí permite la posibilidad de relacionar dos asociaciones con objetivos opuestos (atributo *EOpposite*), lo cual es una forma equivalente de representar la misma asociación. De esta forma queda especificada sin ambigüedades la navegabilidad de toda asociación en el diagrama.
- Como consecuencia de la limitación anterior, con EMF no se puede expresar la cardinalidad origen de las asociaciones, únicamente se puede expresar la cardinalidad destino. En el caso de una asociación bidireccional es preciso definir dos cardinalidades.
- EMF no dispone de agregaciones, aunque sí permite la asociación normal entre dos clases (con navegabilidad definida) y la relación de composición. Esta última es de vital importancia puesto que permite crear objetos dinámicamente, fijando qué clase tiene potestad para hacerlo.
- Dado que los objetos en XMI se representan en un árbol de agregaciones, un elemento contenido o anidado no puede pertenecer al mismo tiempo a dos ramas diferentes que contengan una agregación del mismo. Dicha situación causaría una ambigüedad. En tal caso sólo una de las dos ramas debe tener la composición, obligando al otro objeto contenedor a que forme como asociación, de manera que uno de los objetos contenga el objeto dentro de su rama y el otro mantenga una referencia al objeto. Otra solución es que ninguno de los dos contenedores contenga el objeto y todo se gestione mediante referencias al mismo, pero normalmente el objeto contenedor más relacionado es el que finalmente contiene el objeto.

Estas limitaciones de EMF obligan a adaptar los metamodelos MMD y MMreq a EMF, lo que implica una adaptación del modelo PIM de MMD y MMreq (*Platform Independent Model*, modelo independiente de plataforma, especificado en MOF en secciones anteriores) al modelo PSM de MMD.ecore y MMreq.ecore (*Platform Specific Model*, modelo específico de plataforma, especificado en Ecore).

4.4.4 MMD-FEAT adaptado a EMF

En esta sección presentamos el metamodelo *MMD-FEAT.ecore* (Figura 46) que se crea a partir de la adaptación a EMF/Ecore de MMD-FEAT a nivel conceptual (Figura 39 en

la página 251). En la Figura 46 se puede observar como la clase *Repository* contiene una composición de *Catalogue*, siendo esta clase la encargada de guardar las colecciones de los elementos reutilizables del modelo compuesto por las colecciones de *dataTypes*, *parameters*, *textualRequirements* y *features*.

La imposibilidad de discernir entre agregaciones y composiciones colisiona con el modelo conceptual MMD-FEAT; debemos elegir qué clases deben mantener la capacidad de crear instancias. En MMD-FEAT, en el caso de la clase *Feature* este conflicto aparece en varias ocasiones:

- Entre *Catalogue* y la propia *Feature,* ya que ambas tienen una composición de *Feature* (una mantiene una relación con todas las características del catálogo y la otra sólo con las características que la refinan). La solución por la que se ha optado en MMD-FEAT.ecore es que las características estén organizadas en el catálogo a través de una relación de composición, y que la agregación *refine-and* se convierta en una relación de asociación. Indirectamente se sigue manteniendo la estructura de árbol, por lo que es una solución sintácticamente equivalente a la original.
- En la agregación que mantiene un punto de variación con sus variantes (*Vp_Feature* y *Variants*), ya que por medio de la especialización, la potestad de crear *features* la mantiene la clase *Catalogue.*
- En las relaciones de composición *favours, conditions, requires* y *excludes*. Al ser destino y origen de las mismas una *feature* las convertimos en asociaciones pues la clase *feature* ya está contenida por *Catalogue*. Podemos considerar también *refine-and* y *variants* como dos relaciones más de composición, por lo cual se les aplica la misma solución.
- En las agregaciones de *Parameter* desde *ParametrizedFeature*, así como de *InstantiatedParameter* a partir de *InstantiatedParametrizedFeature*. La primera agregación no puede convertirse en una relación de composición porque entra en conflicto la reutilización de parámetros desde el catálogo con la posesión de parámetros desde una característica. Dado que los parámetros son un elemento importante de reutilización del catálogo, se prefiere mantener la composición desde *Catalogue*. Los parámetros instanciados (*InstantiatedParameter*) son exclusivos de las características instanciadas que hacen uso de ellos (*InstantiatedParametrizedFeature*), por lo que esa relación de composición sí podría establecerse. No obstante, debido a otras causas (ver explicación en el siguiente párrafo), finalmente se eliminan del modelo estos dos tipos de especializaciones de *Feature* y son sustituidos por una asociación con *Parameter* de tal forma que toda *feature* pueda ser parametrizada o parametrizada instanciada.

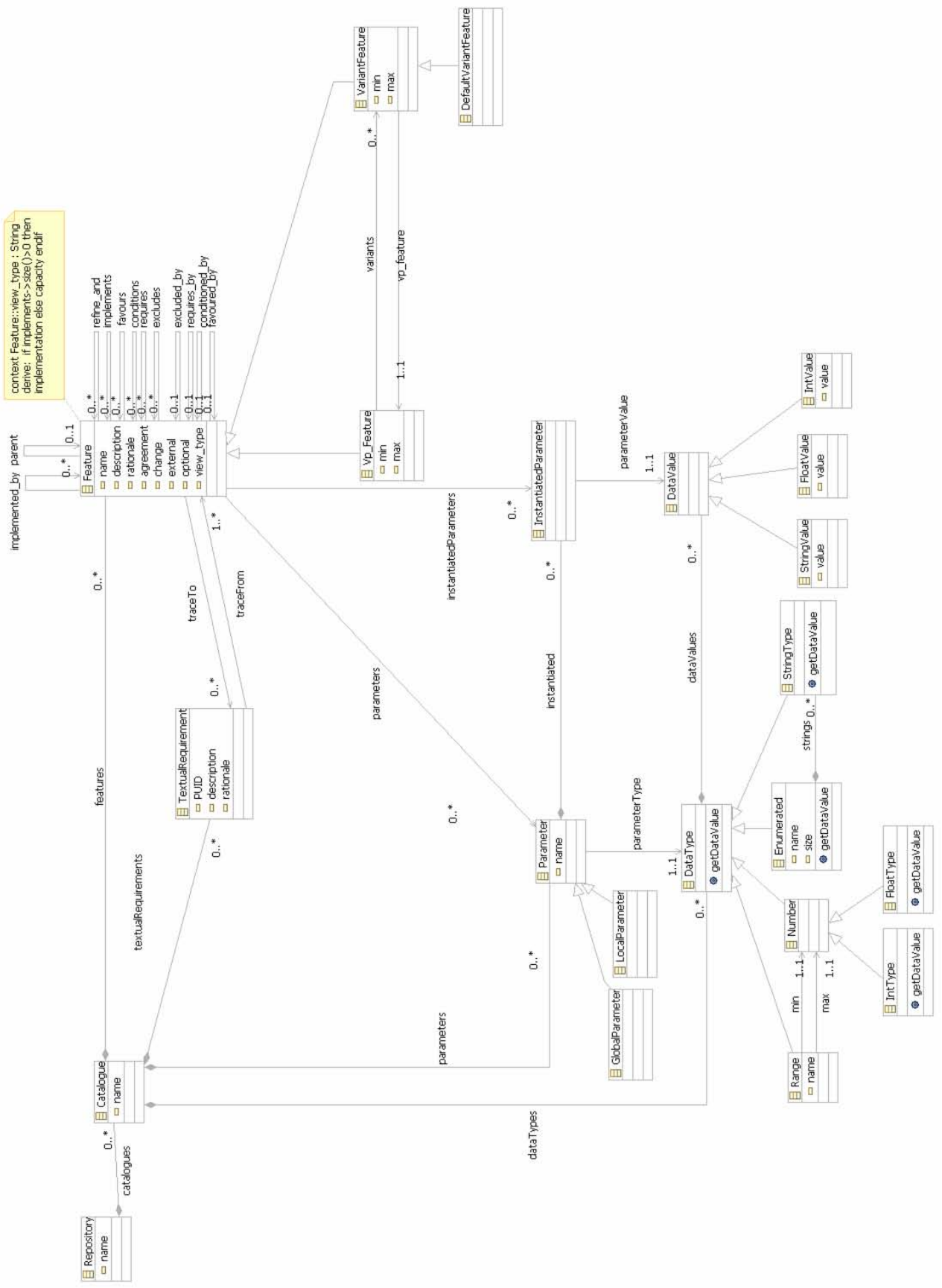

Figura 46 MMD-FEAT.ecore

Tenemos que adaptar la especialización de *Feature* de forma que permita todos los subtipos especificados en MMD. En el diagrama MOF de MMD a nivel conceptual se indica que una característica puede especializarse de diferentes formas:

- Puede especializarse en su parametrización, de forma que puede estar parametrizada o parametrizada instanciada.
- Puede especializarse en el tipo *Vp-Feature*, de forma que represente una variante o un punto de variación.
- Puede especializarse según su nivel de abstracción, de forma que existen las características de implementación y las características de capacidad.

Además, una característica debería poder especializarse en los tres tipos anteriores a la vez, de forma que se pudieran expresar todas las posibles combinaciones. Una solución consiste en añadir una asociación de *Feature* a *Parameter* (y otra con respecto a *InstantiatedParameter*) de tal forma que todas las especializaciones de una característica puedan estar parametrizadas. Se añade un atributo derivado *view_type* que calcula si la característica es de capacidad o de implementación, en función de si su relación *implements* existe (en la Figura 46 este cálculo se especifica en una nota con OCL). Así pues conseguimos implementar los tres tipos de especialización y eliminar uno de los conflictos entre asociaciones y agregaciones.

4.4.5 MMD-UC adaptado a EMF

En esta sección presentamos el metamodelo *MMD-UC.ecore* (Figura 47) que se crea con la adaptación de MMD-UC a nivel conceptual (Figura 40 en la página 254) a EMF/Ecore.

Algunas asociaciones en el nivel conceptual dan lugar a nuevas clases como se comenta a continuación. Una nueva clase en MMD-UC.ecore es *Include*, que sustituye a la asociación con ese mismo nombre en MMD-UC. *Include* contiene un nombre único que la identifica y una serie de relaciones hacia y desde *UseCase* que indican que un caso de uso incluye a otro u otros. Otra clase añadida al metamodelo MMD-UC.ecore es *ImpactLink*. Esta clase relaciona *ChangeCase* con *UseCase* para indicar el impacto que puede tener en un caso de uso la introducción de un futuro caso de uso (*ChangeCase*). Para relacionar un caso de uso con su actor primario se crea la clase *Association*. Para especificar la generalización de actores se crea *Generalization*, que tiene relaciones para indicar que un actor generaliza a otros o que un actor está generalizado por otro.

El hecho de que un paso de caso de uso está contenido en un caso de uso se especifica mediante la relación *ucStepIn*, que indica el caso de uso que contiene este paso. Un paso de un caso de uso puede estar relacionado con algunas características, que se señalan mediante la relación *features*. La clase *Catalogue* introduce ahora, además de los elementos reseñados en MMD-FEAT.ecore, colecciones de *Include*, *impactLink*, *Association*, *Actor* y *Generalization.*

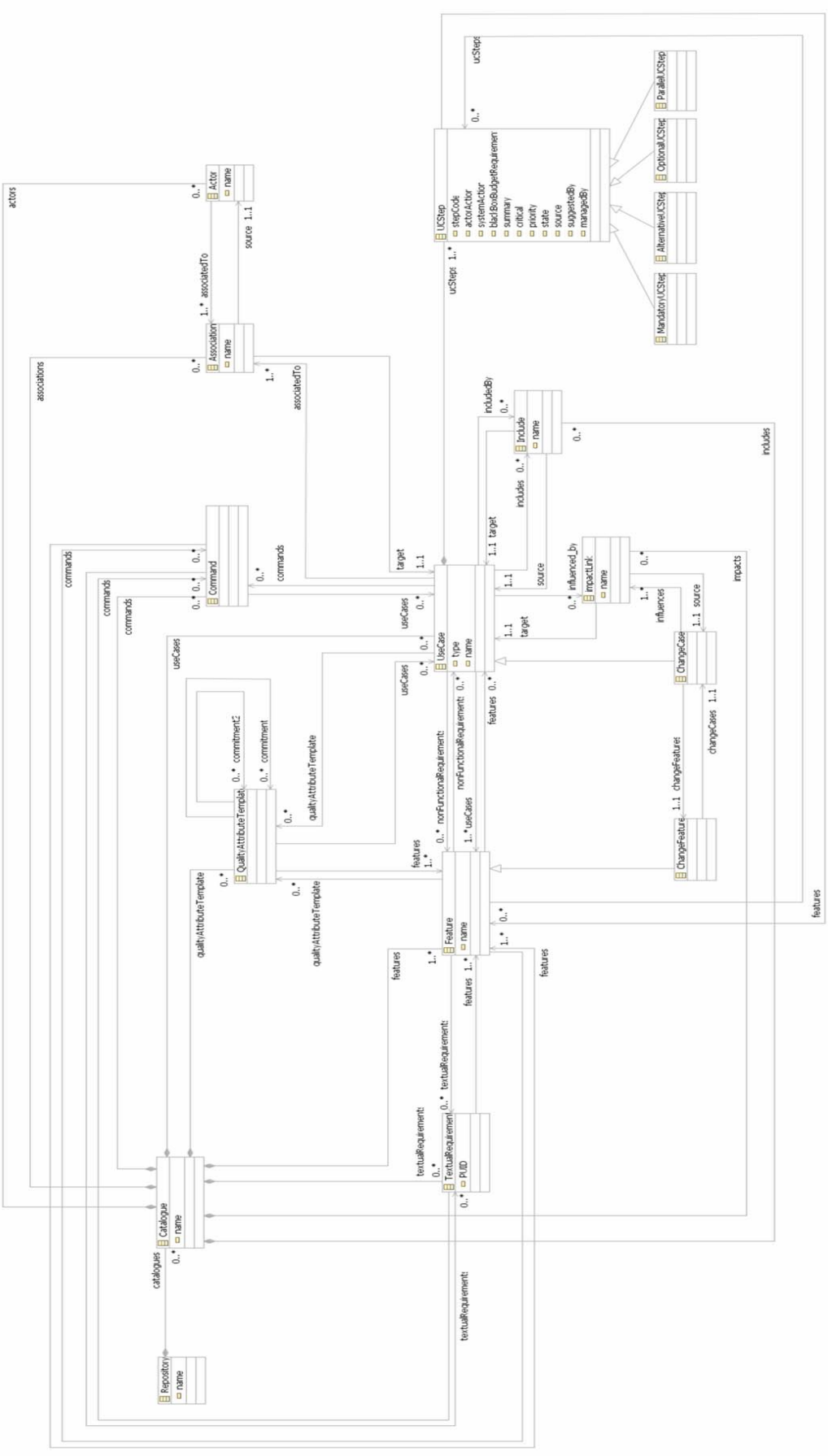

Figura 47 MMD-UC.ecore

4.4.6 MMreq adaptado a EMF

En esta sección presentamos *MMreq.ecore* (Figura 48), es decir, el metamodelo de requisitos MMreq resultado de adaptar a EMF/Ecore el metamodelo conceptual MMreq (Figura 41 en la página 257). En la Figura 48 se puede observar como un repositorio (*Repository*) contiene una composición de catálogos (*Catalogue*) que contienen la información reutilizable. En el repositorio se mantiene un contador *nRequirement* que sirve para registrar los identificadores de requisito generados (PUID). Por lo demás *Catalogue* contiene colecciones similares a su homólogo en el dominio MMD-FEAT (Figura 46): *traces*, *documents*, *requirements*, *stakeholders*, *requirementsTypes*, *parameters* y *dataTypes.*

Al igual que en la adaptación de MMD a EMF, en la adaptación de MMreq encontramos conflictos de agregaciones y composiciones:

- En MMreq se especifica un documento (*Document*) como una composición de secciones (*Section*), pero además existe una asociación reflexiva en *Section* que especifica las subsecciones, por lo cual aparece conflicto. Se ha decidido que sea el documento el que mantenga la relación de composición, como si fuera un repositorio de secciones.
- En MMreq se especifica que una sección está compuesta por una colección de requisitos, que constituyen uno de los elementos reutilizables del catálogo, por lo que sustituimos la relación de composición en *Section* por una asociación.
- En *Parameter* e *InstantiatedParameter* aparece el mismo conflicto que describimos en MMD, y se vuelve a preferir que *Catalogue* mantenga una composición de *Parameter* y que *Parameter* mantenga la relación de *InstantiatedParameter*.

Dado que el catálogo de requisitos no tiene estructura subyacente (a diferencia de los diagramas de características, que tienen estructura de árbol), el hecho de poder acceder a los requisitos mediante relaciones de traza (dado que la navegabilidad es en ambos sentidos) enriquece el modelo añadiendo posibles vías de organización o vistas diferentes de los requisitos. Junto con el catálogo de requisitos también se genera un documento de requisitos como vista alternativa que los organiza y agrupa en secciones.

4.4.7 Relaciones de transformación

El desarrollo de las transformaciones ha sido realizado a través de un *script* de transformaciones en lenguaje QVT-*relations*. El método seguido para diseñar las transformaciones se describe pormenorizadamente en el proyecto fin de carrera de Jorge Hernández [135]. En resumen, este método ha sido el siguiente:

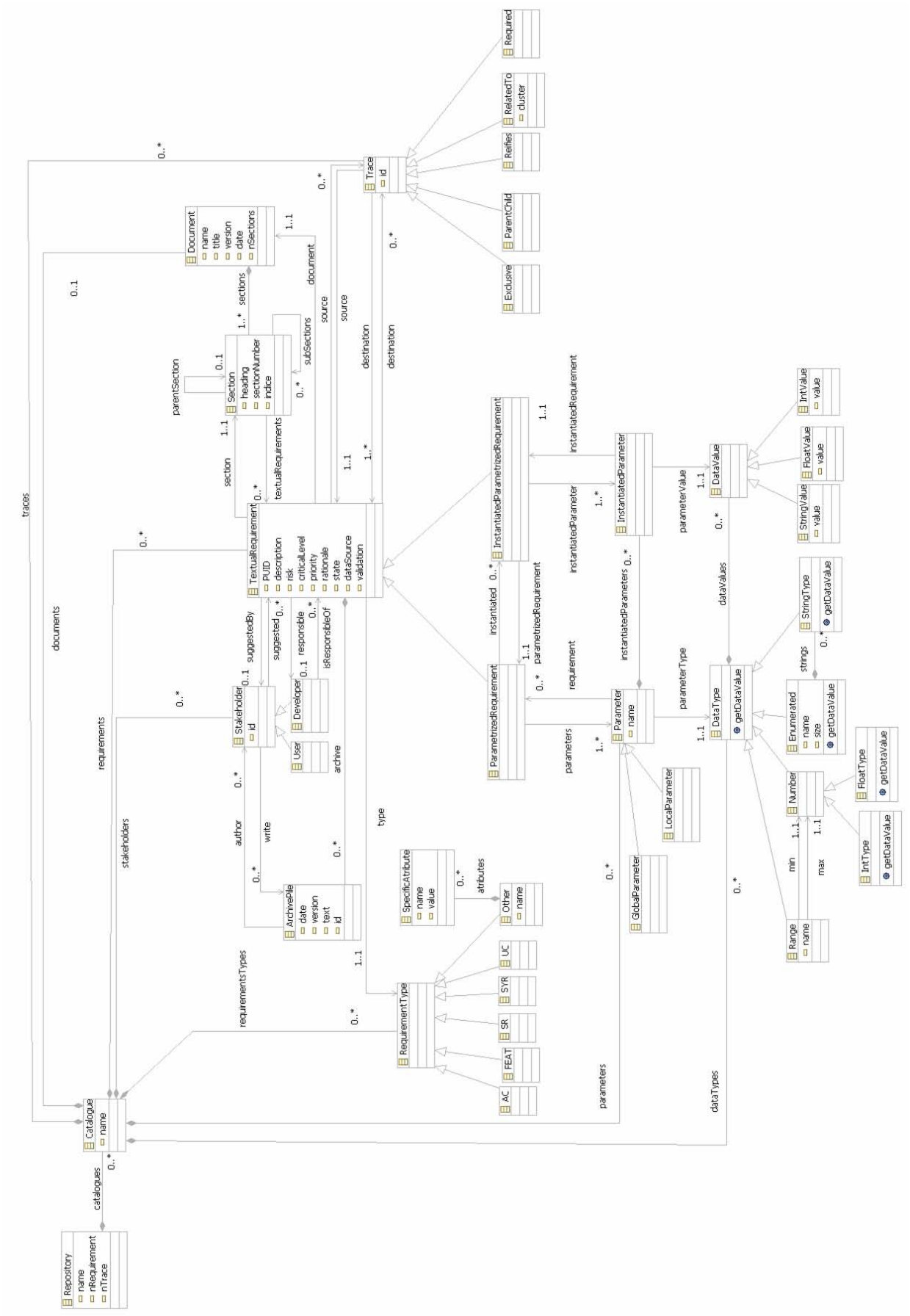

Figura 48 MMreq.ecore

- En primer lugar se han identificado las correspondencias (*matching*) entre objetos equivalentes, a partir de la aplicación del patrón *node mapping* (*correspondencia de nodos*). Este patrón consiste en realizar una identificación de elementos uno a uno. Los elementos identificados son los que tienen una correspondencia directa o casi directa en ambos diagramas. En este contexto entendemos por *elementos* los objetos de las clases definidas en ambos metamodelos. El objetivo es conseguir una "sopa de objetos flotando" de forma que sólo falte el pegamento o unión (*binding*) para unirlos. Cada elemento es captado mediante un patrón que normalmente consiste en la especificación de la clase del objeto que queremos captar y su contexto, lo cual se realiza en QVT mediante una plantilla *checkonly* del objeto y las precondiciones necesarias para ejecutar la regla. La plantilla de creación de este patrón consiste en el objeto objetivo y su contexto, es decir, la plantilla *enforce* del objeto objetivo. Para realizar la aplicación sistemática del patrón se ha realizado un enfoque *top-down*, de forma que primero se identifican los objetos más altos en la jerarquía de especialización y en la jerarquía de composición (los que agregan colecciones de objetos), descendiendo progresivamente a los objetos más simples o menos relacionados (nodos hoja).
- En segundo lugar se han identificado las asociaciones que unen los objetos, de forma que se crean las ligaduras (*bindings*) de las relaciones. Realizamos una aplicación del patrón *relationship mapping* (*correspondencia de relaciones*), de forma que utilizando los elementos identificados anteriormente con el patrón *node mapping* y sus asociaciones, encontramos patrones *relationship* para traducirlos al nuevo modelo. La plantilla QVT consiste en la identificación de los dos elementos que forman la asociación en su contexto, la estructura de la relación captada se refleja en el *matching* y la relación creada en la plantilla de creación. Para expresar el contexto puede utilizarse la cláusula *When* y para la creación de objetos puede utilizarse la cláusula *Where*, siendo ambas opcionales en función del estilo de programación de las plantillas.

La aplicación sistemática de estos patrones ha sido realizada por medio del estudio de distintas *zonas* de los modelos en función de su propósito. En este trabajo distinguimos tres zonas de aplicación del metamodelo de origen:

- Jerarquía *Repository–Catalogue–Document*, con el propósito de servir de contexto o marco de las relaciones, puesto que estos objetos van a ser frecuentemente referenciados por los objetos hoja en la jerarquía (nos referimos a este conjunto de relaciones como *Zona 1*).
- Jerarquía *Feature*, con el propósito de creación de objetos de tipo *TextualRequirement* (*Zona 2*).
- Relaciones entre *Feature y TextualRequirement*, con el propósito de la creación de las relaciones que tienen que ver con estos elementos (*Zona 3*). Se divide a su vez en dos zonas: parametrización (*Zona 3.1*) y relaciones de composición (*Zona 3.2*).

Identificada una zona y realizada la correspondencia entre nodos, se identifican las asociaciones existentes en el modelo fuente y las implicaciones de creación en el modelo destino, creando para cada una de ellas las reglas QVT-*relations* adecuadas. En las siguientes secciones describimos las correspondencias QVT-*relations*. Para cada una

de las transformaciones codificadas se muestra una descripción de la misma en notación gráfica QVT, de forma que las transformaciones sean comprensibles más fácilmente. Para no extender demasiado la explicación, en las siguientes secciones repasamos sólo las transformaciones más importantes entre MMD-FEAT y MMreq. En los proyectos fin de carrera de Jorge Hernández [135] y Ana Pla [248] se puede encontrar el código completo de las transformaciones desde MMD-FEAT.ecore y MMD-UC.ecore a MMreq.ecore (40 transformaciones en el primer proyecto y 56 en el segundo).

La propuesta de notación gráfica para QVT [235] nace como una extensión de los *script* de transformación con la intención de mostrar de forma sencilla la regla de transformación. Generalmente se representa a la izquierda el patrón que se busca en el modelo fuente y a la derecha el patrón de objetos que se genera. Una transformación puede relacionar dos o más patrones, siendo un patrón una colección de objetos, enlaces y valores. La estructura del patrón se define por objetos y enlaces entre ellos y puede ser representada también por diagramas de objetos UML. La notación definida en el estándar se basa en estos diagramas con las siguientes extensiones:

- Se añade un símbolo especial que representa la transformación. El símbolo es un hexágono del que brotan tantas flechas como patrones haya involucrados.
- Encima de cada flecha que apunte a un patrón, se coloca el metamodelo al que hace referencia y el tipo del patrón. Puede ser de tipo C (*Checkonly* o sólo lectura) o de tipo E (*Enforceable*, obligatoria, el patrón está obligado a cumplirla aunque implique la creación o eliminación de objetos).
- Con el texto <<*domain*>> sobre un objeto se indica el dominio del patrón, o el objeto raíz por donde el motor de QVT debe empezar a buscar el patrón.
- En dos notas aparte del diagrama se indican las cláusulas de precondición o postcondición de la realización de la transformación. En el contexto de QVT-*relations* se denominan *When* y *Where*, respectivamente.

Si bien los diagramas de transformación son autodescriptivos y fácilmente comprensibles, en algunos puntos la representación gráfica puede volverse demasiado farragosa. Por ejemplo, si la regla envuelve muchos dominios o si la estructura del patrón resultante está muy relacionada entre los objetos (con varias relaciones entre los mismos objetos). Para ese tipo situaciones aumentamos la notación para que incluya la máxima información posible, como los nombres de las asociaciones o la inclusión del propio código QVT.

4.4.7.1 Jerarquía *Repository-Catalogue-Document*

A través del estudio de los diagramas MMD-FEAT.ecore y MMreq.ecore se puede observar que tanto *Catalogue* como *Repository* son comunes en ambos modelos. En esencia una *feature* (su campo *description*) se traduce directamente a un *textualRequirement*, pero para realizar esta transformación es necesario tener un marco común. La relación con los catálogos y con el documento va a permitir alojar las características en la sección, documento, catálogo y repositorio adecuados.

En la Figura 49 se muestra la correspondencia entre objetos *Repository*, un ejemplo claro del patrón *node mapping*, en la cual el contexto es vacío, por lo que no tiene precondiciones.

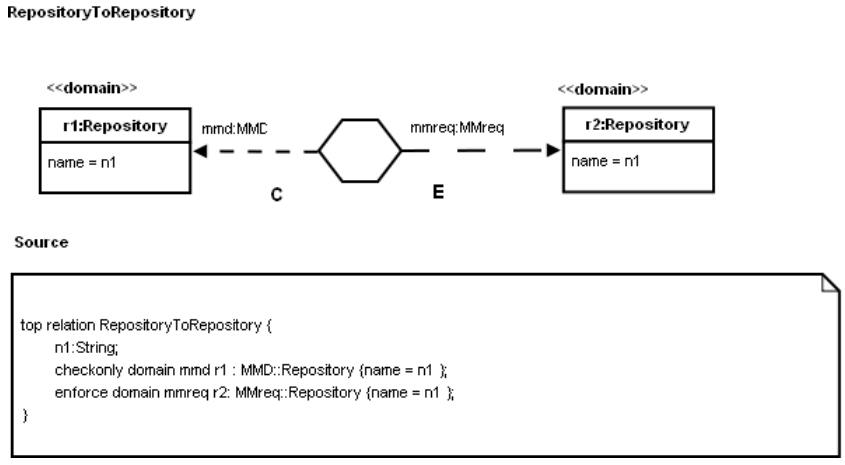

Figura 49 *RepositoryToRepository*

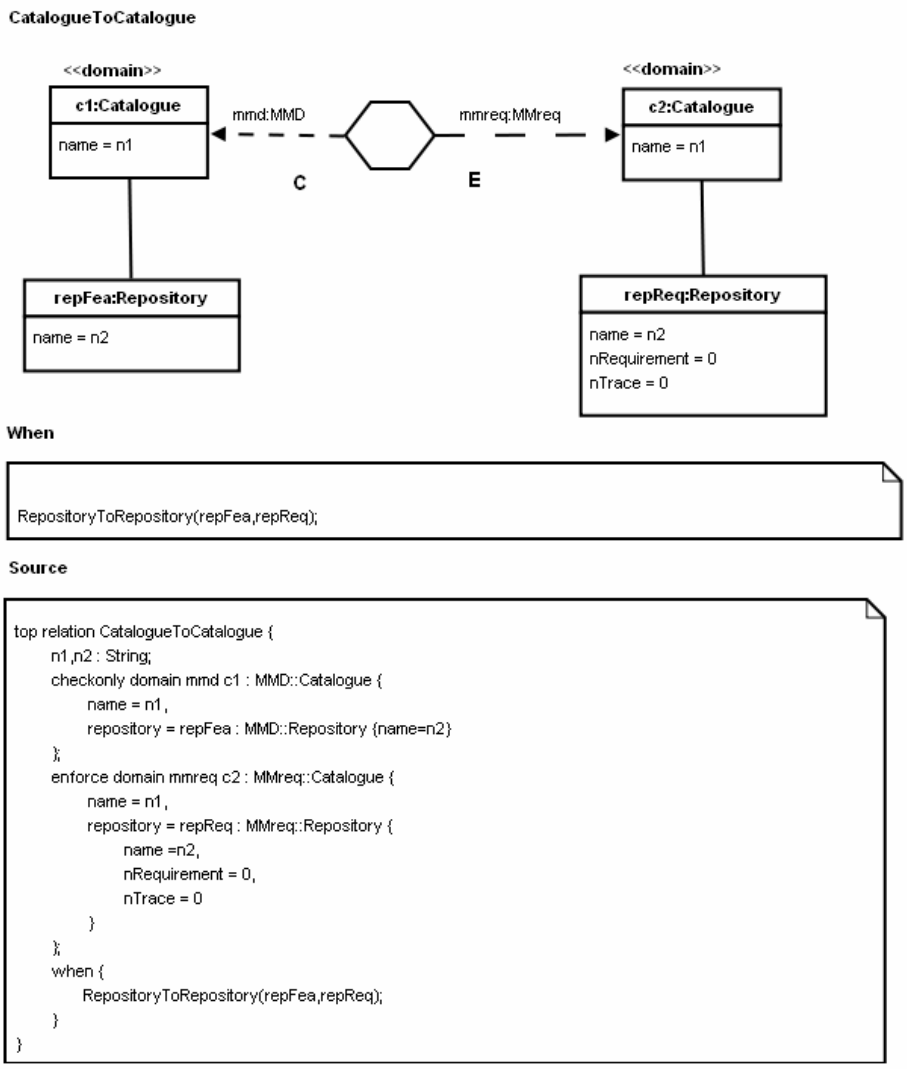

Figura 50 *CatalogueToCatalogue*

En la Figura 50 se muestra la correspondencia entre objetos *Catalogue*, de nuevo un ejemplo del patrón *node mapping*, pero en este caso en su contexto tiene en cuenta que

el catálogo debe pertenecer a un repositorio tanto en el modelo fuente como en el destino. Aprovechamos la creación del objeto *catalogue* para poner a cero los contadores numéricos (*nRequirement* y *nTraces*) que servirán para generar los identificadores de objeto.

En la Figura 51 se crea un objeto de tipo *Document* por cada catálogo definido en el modelo de origen MMD. Nos encontramos de nuevo ante un ejemplo del patrón *node mapping*. En este caso el contexto viene dado porque la regla *CatalogueToCatalogue* haya sido disparada, lo que a su vez implica que la relación entre repositorios también haya sido disparada. En el modelo destino MMreq, un documento sólo es alojado en su catálogo que a su vez está alojado en su Repositorio, de ahí el nombre de jerarquía *Repository–Catalogue–Document*.

La asociación que liga un *catalogue* con una *feature* (*features* en MMD-FEAT.ecore) marca la referencia de la característica a su contenedor, de forma que pasando un *catalogue* como parámetro estamos pasando la colección de características, lo cual nos será muy útil en las siguientes transformaciones para operar sobre un conjunto de características.

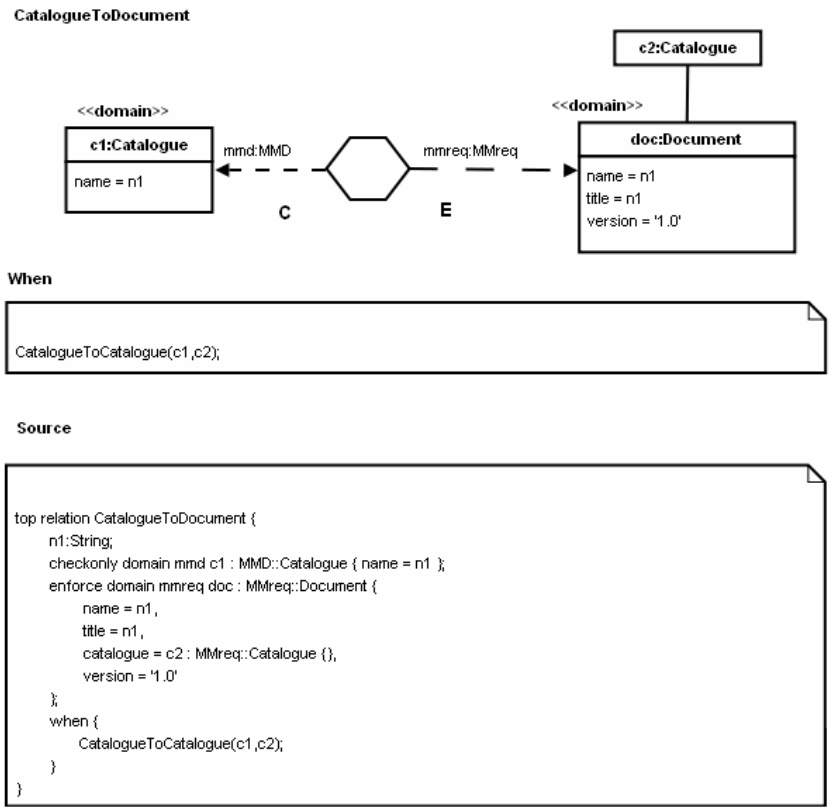

Figura 51 *CatalogueToDocument*

En la especificación de QVT encontramos la definición de *clave*, *key*, como aquel conjunto de propiedades de una clase MOF que pueden identificar unívocamente una instancia de la clase. Las propiedades que definen la *key* son llamadas *propiedades identificativas*, y se utilizan en las plantillas de *matching* (búsqueda de patrones) para determinar cuántas instancias de una clase son candidatas a ejecutar la regla. En las plantillas de creación las *keys* se utilizan para determinar cuántas instancias son

candidatas a ejecutar la regla o deberían existir en una relación por medio de la creación. Una *key* puede ser derivada desde los valores basados en las propiedades identificativas, por lo que no es necesariamente un valor individual. Desde el punto de vista sintáctico, definir los identificadores correctos de cada objeto permite que dos relaciones diferentes sumen la combinación de sus efectos si el destino es el mismo objeto. Los identificadores de los objetos no son elegidos por el autor de la transformación sino que deben venir indicados en la definición del modelo. Los identificadores de la Zona 1 se reflejan en la Tabla 50.

```
key MMD::Catalogue {name};
key MMD::Repository {name};

key MMreq::Catalogue{name};
key MMreq::Document {name,catalogue};
key MMreq::Repository {name};
```

Tabla 50 Identificadores QVT de la Zona 1

La Tabla 50 implica que los objetos catálogo y repositorio se identifican mediante su nombre, de forma que no puede haber ni dos catálogos con el mismo nombre, ni dos repositorios con el mismo nombre. Sin embargo, los objetos de tipo documento van a ser identificados por la referencia al objeto catálogo al que pertenecen, de forma que se permite la existencia de dos objetos documento con el mismo nombre si están en diferentes catálogos.

4.4.7.2 Jerarquía *Feature*

En esta jerarquía tenemos en cuenta que en el metamodelo de origen, MMD-FEAT.ecore, una característica puede estar compuesta por un conjunto de *subcaracterísticas* como indica la relación *refine-and*. Análogamente, en el metamodelo destino MMreq, un requisito que provenga de una característica que se refine en varias debe agrupar todos sus nuevos requisitos hijos en la misma sección del documento. Además este nuevo requisito en MMreq mantiene la relación con requisitos derivados de sus subcaracterísticas por medio de relaciones de traza, como se verá en la sección de relaciones entre *Feature y Requirement* (Sección 4.4.7.3). Para llevar a buen término esta relación *FeaturesToRequirements* se identifica primero la correspondencia entre *features* y *requirements*, y continuación se trata el caso de una característica que se refina y otra que no.

En la Figura 52 se muestra la correspondencia entre una *feature* y un *textualRequirement*, la regla se dispara ante un objeto de tipo *Feature* alojado en un catálogo. El contexto fuerza a que esta regla sólo se ejecute después de que se hayan ejecutado las reglas referentes a los subtipos de *TextualRequirement*, ya que la primera regla en ejecutar es la que crea los objetos. Si el orden de ejecución de las reglas fuera el contrario no se crearía objetos de los subtipos especializados, sino sólo de los supertipos. La regla sólo se dispara si antes se ha ejecutado la regla referente a características parametrizadas (instanciadas o no) y características normales (sin especializar). La ejecución de estas reglas determina la creación de un requisito

parametrizado o de un requisito parametrizado instanciado o de un requisito sin especializar.

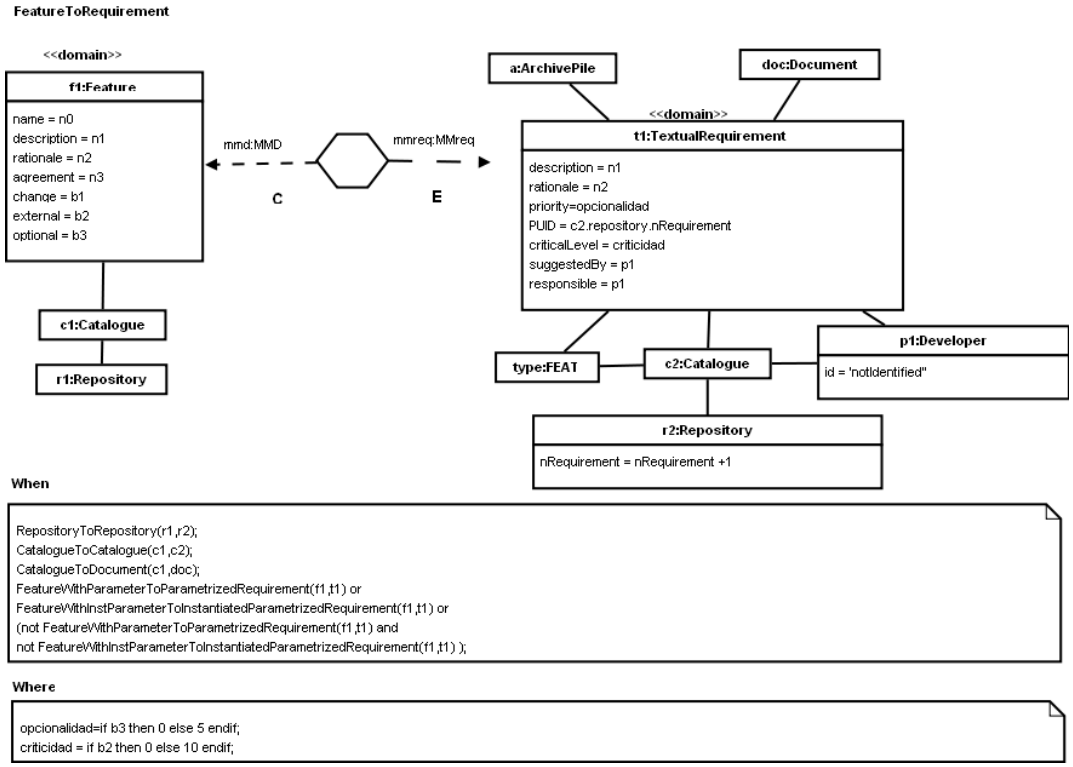

Figura 52 *FeatureToRequirement*

La regla es de nuevo un ejemplo del patrón *node mapping*, donde el contexto viene determinado por la herencia del objeto *textualRequirement* y por el contexto de *repository-catalogue-document*. La plantilla de creación del requisito toma la información suministrada por la característica de forma que:

- El campo *PUID* (*Project Unique IDentification, Identificador del requisito*) toma el valor apuntado por *nRequirement* en el repositorio de MMreq.ecore. En la misma plantilla de creación del objeto podemos incrementar este valor justo después de tomar la copia.
- El campo *description* del requisito debe tener como valor inicial el campo *description* del objeto *feature.*
- El campo *rationale* del requisito debe ser el del campo homónimo del objeto *feature.*
- El campo *priority* toma el valor 0 si proviene de una característica opcional, 5 si proviene de una característica no opcional.
- El campo *criticalLevel* toma el valor 0 (bajo) si proviene de una característica externa, pues no necesita ser implementado; 10 (alto) si es de tipo obligatorio.
- Los campos *suggestedBy* y *responsible* se rellenan con un objeto tipo persona sin identificar, llamada '*notIdentified'*, de forma que todos los requisitos no asignados a ningún interesado son asignados a este sujeto especial.

- En el campo *dataSource* (fuente del requisito), a los requisitos extraídos a partir de *ExternalFeature* se les asigna el valor "external", mientras que a los que proceden de *ChangeFeature* se les asigna el valor "change". En otro caso se deja el campo en blanco puesto que se ignora su procedencia.
- Se le asocia un objeto *archivePile* en el que se anota el historial de revisiones del texto del requisito.
- Se le asocia el tipo de requisito *FEAT* por ser generado a partir de una característica.

A continuación suministramos información sobre las asociaciones creadas:

- *catalogue*: referencia al objeto catálogo donde se encuentra ubicado el requisito.
- *destination*: referencias a los objetos traza donde figura como destino de las mismas (ver reglas generadoras de trazas como *RelationFeatureImplements*).
- *document:* referencia al objeto documento que contiene el requisito (ver *CatalogueToDocument*).
- *section*: referencia al objeto *Section* del documento donde está alojado (ver reglas de generación de secciones).
- *source*: colección de referencias a objetos de tipo *Trace* de los cuales parten o tienen su origen en este requisito (ver reglas de generación de trazas).

En la relación de la Figura 53 se contempla la creación de una sección especial con el encabezado '*root_undf*', que contendrá los requisitos agrupados en la sección principal del documento. La relación sólo debe dispararse si no se cumple la relación *FeaturesToSections* (Figura 54), es decir si la característica que lo dispara es huérfana y por tanto no pertenece a ninguna sección del documento. Por ello los requisitos que provengan de características huérfanas o raíz se asocian a la sección especial '*root_undf*'.

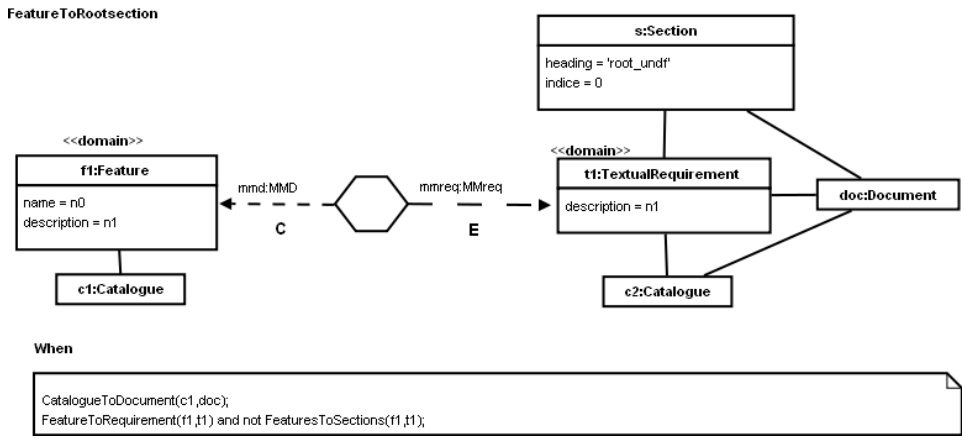

Figura 53 *FeatureToRootSection*

La transformación *FeaturesToSections* (Figura 54) implica la creación de una sección con el nombre de la característica padre, donde se agrupan todos los requisitos derivados de sus características hijas. Para la ejecución de esta regla se toma como

precondición una nota OCL que indica que el padre de la característica no debe ser indefinido. Además para la ejecución correcta de esta regla se asume que tanto el contexto de catálogos como el requisito ya han sido creados. Esta regla se ejecuta para cada uno de los hijos de una característica, y gracias al identificador de sección, *heading*, fijado con la cláusula *Key* (Tabla 51), no se crean nuevas secciones para cada hijo sino que todos los hijos de una característica van a parar a la misma sección del documento.

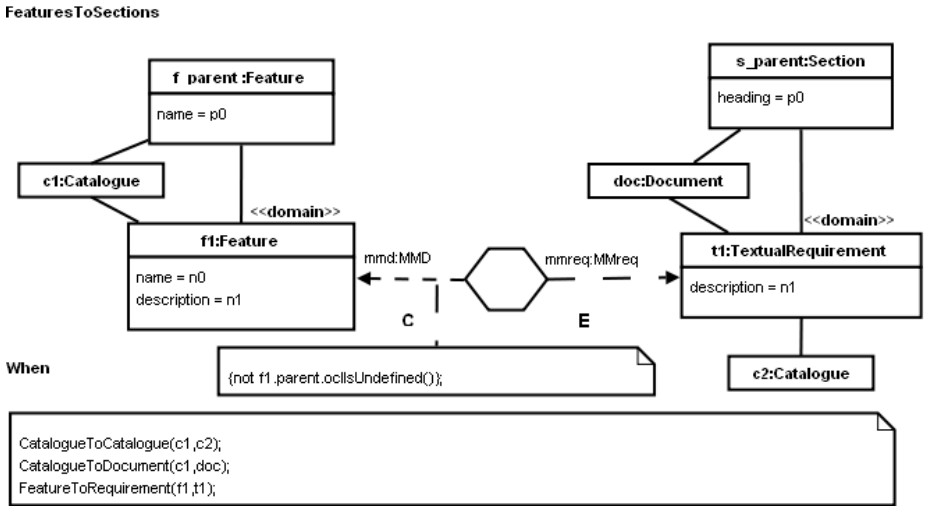

Figura 54 *FeaturesToSections*

En la relación *FeatureToSubsections* (Figura 55) se realiza la identificación de una jerarquía entre objetos de tipo *Feature* (*f1, f_parent1, f_parent2*) unidos por la asociación *parent* en el dominio MMD-FEAT.ecore. A tal patrón de *Feature* le corresponde una jerarquía de secciones (*s1_parent1* y *s1*) en el dominio MMreq.ecore. Esta regla es imprescindible para creación del valor *sectionNumber* a través de la postcondición *'calculo'* que tiene en cuenta los encabezados de la jerarquía; además con cada ejecución de la regla se actualiza el número de subsecciones que tiene la sección padre (*s1_parent1*).

En la relación *FeatureToRootSubsection* (Figura 56) se muestra en el dominio de MMD el patrón de feature y feature refinada; al trasladarse al dominio MMreq este patrón implica la creación de dos requisitos, cada uno de los cuales tendrá su sección. Dichos requisitos ya deben existir por lo que se reutiliza la plantilla de creación que une dos *textualRequirement*, reutilizando en el contexto la regla *FeatureToRequirement* (Figura 52) como precondición de forma que no se crean nuevos objetos sino que son ligados los existentes. Esta regla es necesaria para el cálculo correcto en el caso base de *parentSection* y *sectionNumber*, pues entre las precondiciones está el haberse ejecutado *FeatureToRootsection* (Figura 53), lo cual indica que la *feature* padre es huérfana.

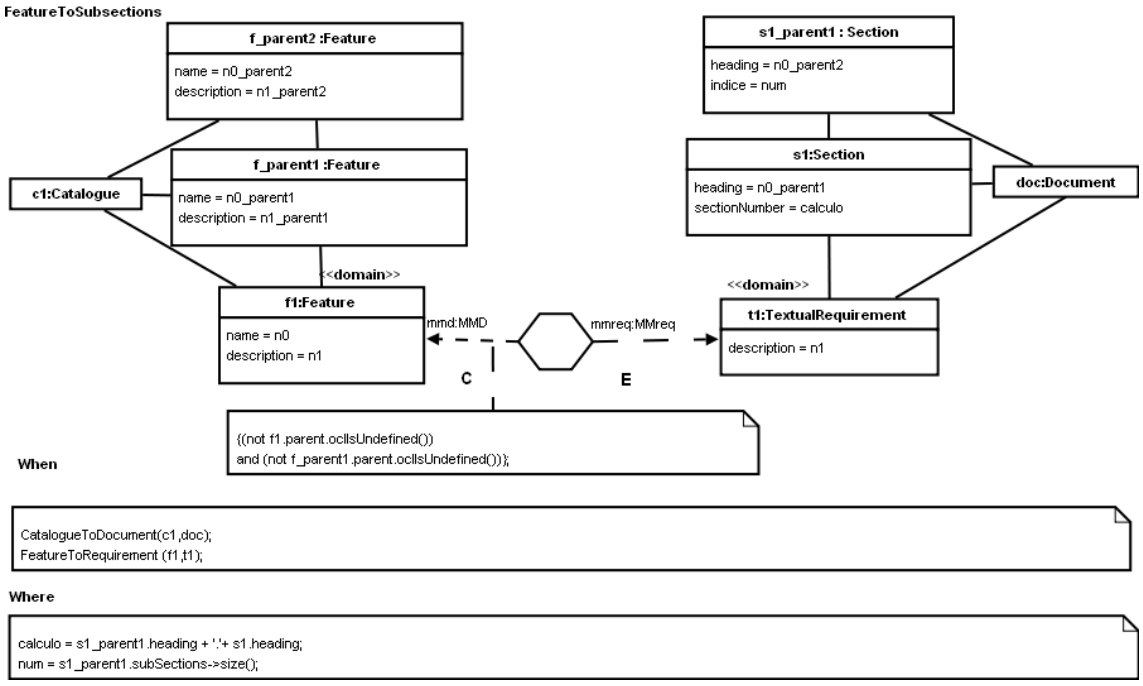

Figura 55 *FeatureToSubsections*

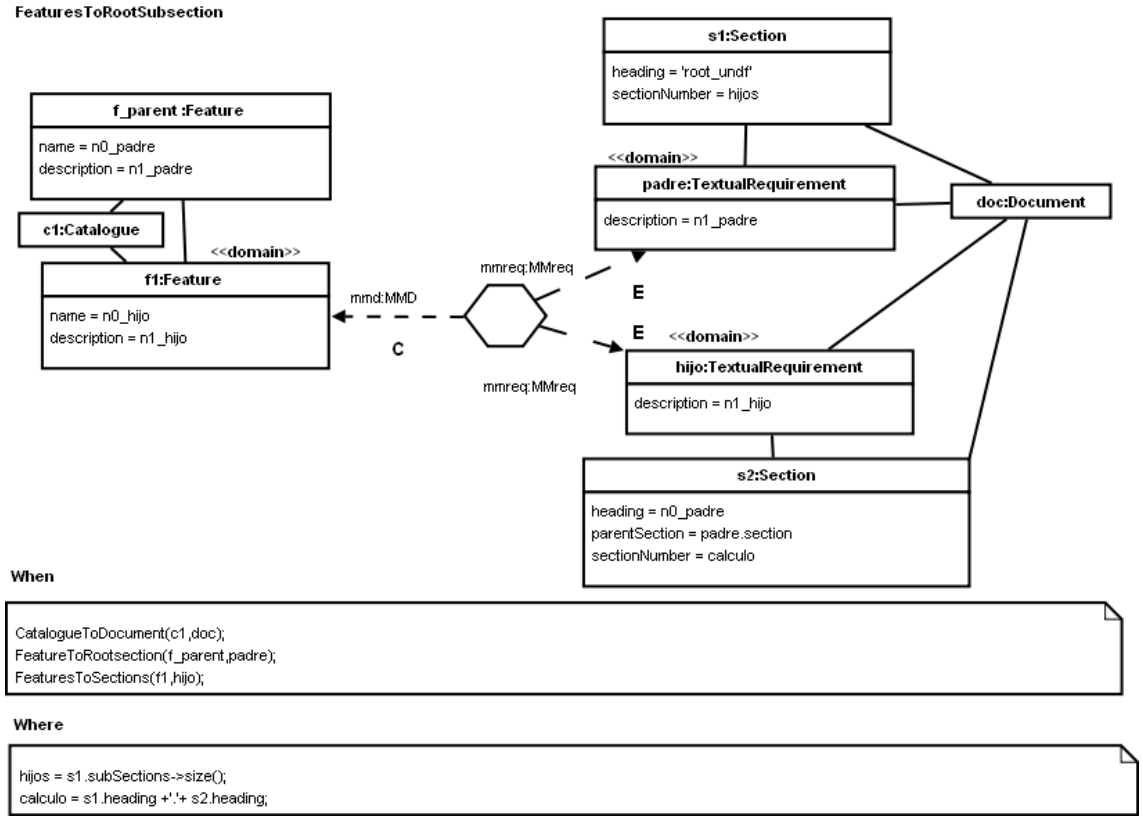

Figura 56 *FeatureToRootsubsection*

Las claves relacionadas con los objetos de esta sección vienen determinadas por la Tabla 51. Se permiten *features* con el mismo nombre en distintos catálogos, de tal forma que el catálogo se vuelve imprescindible para la identificación de la característica. La identificación de las características por catálogo más nombre es

necesaria para permitir un requisito parametrizado y un requisito parametrizado instanciado que puedan tener el mismo nombre estando en diferentes catálogos. Los objetos de tipo *RequirementType* se identifican por catálogo, sólo puede existir uno de cada tipo por catálogo, lo que tiene como resultado que no se multipliquen los objetos de tipo *RequirementType* en cada ejecución de la relación *FeatureToRequirement*. Las secciones se identifican por el documento en el que se encuentran, pudiéndose repetir su nombre en diferentes documentos. Los requisitos textuales son identificados por el *PUID*, que viene determinado por la variable alojada en el repositorio *nRequirement*, por lo que su identificador será único para todo el repositorio.

key MMD::Feature {name,catalogue};
key MMreq::ArchivePile {id};
key MMreq::RequirementType {catalogue};
key MMreq::Section {heading,document};
key MMreq::Stakeholder {id,catalogue};
key MMreq::TextualRequirement {PUID};

Tabla 51 Identificadores QVT de la Zona 2

4.4.7.3 Relaciones entre *Feature* y *Requirement*

Relación de parametrización

En MMD-FEAT.ecore (Figura 46) una característica parametrizada tiene asociada una variable de tipo parámetro (*GlobalParameteter* o *LocalParameter*). Una característica parametrizada instanciada tiene asociado un parámetro instanciado (*InstantiatedParameter*). En este contexto se define la relación de parametrización. Este conjunto de asociaciones debe trasladarse al dominio de requisitos MMreq.ecore, en el cual existen subclases específicas de requisitos parametrizados (*ParametrizedRequirement* e *InstantiatedParametrizedRequirement*).

En la Figura 57 se describe cómo una característica con un conjunto de parámetros se traduce en un requisito parametrizado con el mismo conjunto de parámetros, donde una relación uno a uno entre cada uno de los parámetros viene dada por el contexto *ParameterToParameter*.

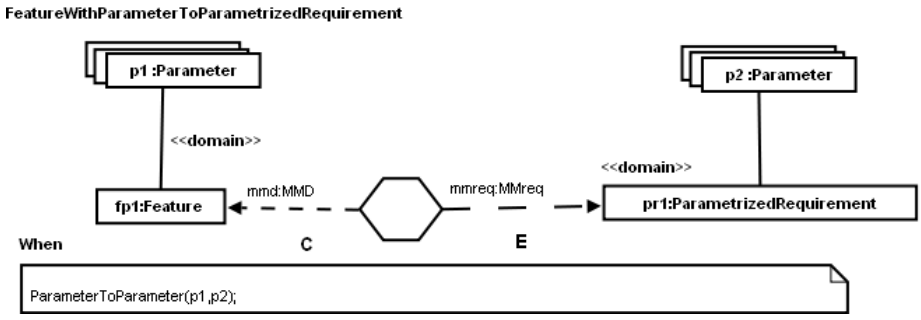

Figura 57 *FeatureWithParameterToParametrizedRequirement*

En la Figura 58 se describe una característica (*fp1*) con un conjunto de parámetros instanciados (*ip1*) que están asociados a un parámetro (*p1*). Dicho patrón se traduce en el modelo destino en un requisito parametrizado instanciado con un conjunto de

parámetros instanciados que cumplen la condición *ConditionParameterToInstantiatedParameter* (Figura 59). Dicha condición comprueba que el parámetro (*p1*, al cual hace referencia el conjunto de características instanciadas *ip1*) tiene su correspondencia en el dominio objetivo MMreq.ecore. Dicha expresión podría haberse expresado en la misma regla dándole un aspecto más simétrico al patrón de creación de la Figura 58; sin embargo esto no es posible porque se ha definido el objeto *Parameter* como abstracto y por tanto no se pueden crear objetos del mismo en ninguno de los dos dominios. Para solucionar este problema se utiliza una regla llamada *condición* porque es de tipo *checkonly-checkonly*, es decir, no crea objetos; sin embargo, entre la precondiciones en la Figura 59 se encuentra la ejecución de la correspondencia entre nodos *parameter* (*ParameterToParameter*), por lo que se asume que el objeto ya ha sido trasladado a MMreq.ecore y por tanto la transformación se ejecuta sin contratiempos.

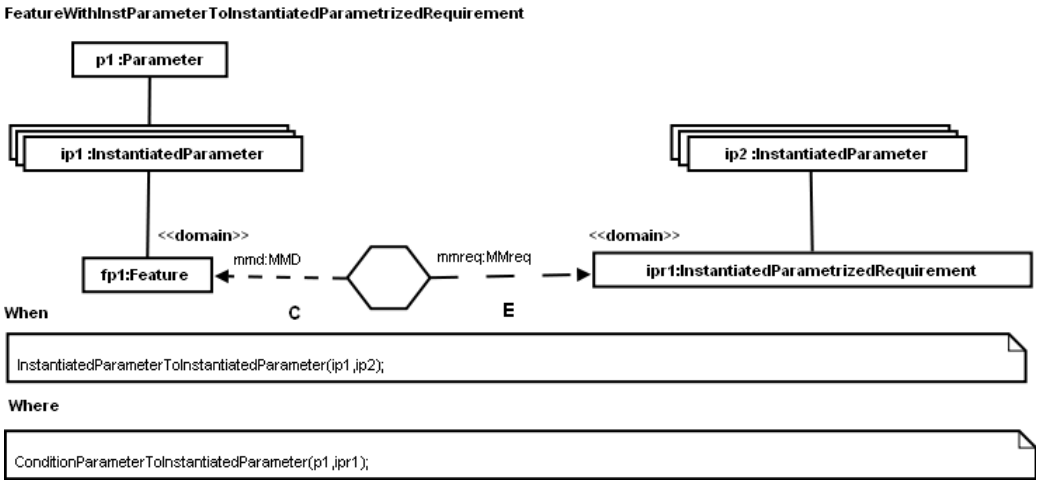

Figura 58 *FeatureWithInstParameterToInstantiatedParametrizedRequirement*

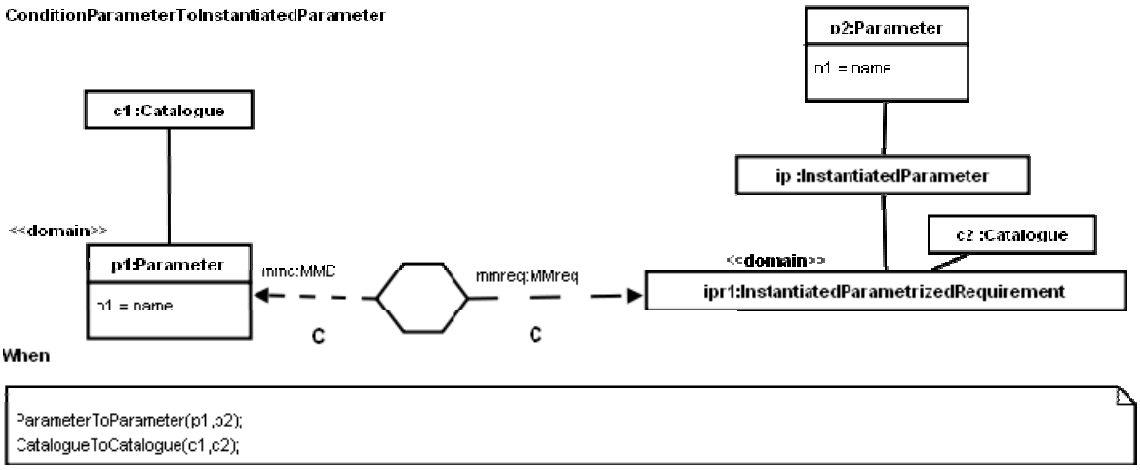

Figura 59 *ConditionParameterToInstantiatedParameter*

En la relación *FeaturesToParametrizedRequirements* (Figura 60) se representa una regla compleja que reutiliza las anteriores. El patrón de correspondencia consiste en una característica parametrizada (*fp1*) y una característica parametrizada instanciada (*fp2*), donde sus parámetros y parámetros instanciados están relacionados entre sí. Esto tiene el efecto en el dominio MMreq.ecore de crear un requisito parametrizado instanciado y

un requisito parametrizado en el cual sus parámetros mantienen sus relaciones. Para asegurar el *binding* exitoso de todos los objetos implicados se utilizan las transformaciones de esta sección para ligarlos correctamente.

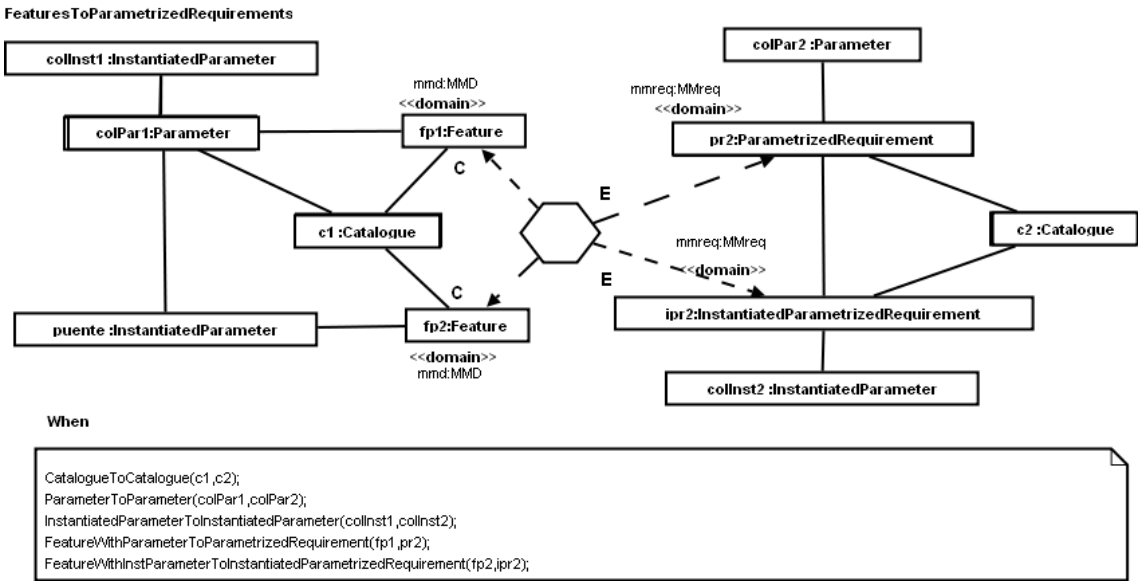

Figura 60 *FeaturesToParametrizedRequirements*

En la Tabla 52 se observan los identificadores de los objetos involucrados en las transformaciones. Un parámetro instanciado se identifica por el parámetro al cual se instancia a un determinado valor. Un parámetro se identifica por el nombre del mismo. Estas claves se mantienen en el destino. En el caso de los *ParameterizedRequirement* e *InstantiatedParameterizedRequirement* se toma el identificador del supertipo, en este caso *PUID*, por lo que no es necesario especificarlo en el *script* QVT.

```
key MMD::Feature {name,catalogue};
key MMD::InstantiatedParameter {parameter};
key MMD::Parameter {name};

key MMreq::TextualRequirement {PUID};
key MMreq::InstantiatedParameter {parameter};
key MMreq::Parameter {name};
```

Tabla 52 Identificadores QVT parametrización

Puntos de variación

Los puntos de variación y la parametrización son fundamentales en la especificación de la variabilidad en los modelos de características, así que es necesario traducir correctamente los requisitos que se corresponden con una característica punto de variación (vp-feature), sus variantes y las relaciones que los unen. Como primer paso se construyen dos correspondencias uno a uno para los nodos *Vp_Feature* y *VariantFeature*, invocando en el contexto *FeatureToRequirement.* Posteriormente se crean dos relaciones entre los hijos del punto de variación.

En la relación *RelationVariantFeature* (Figura 61) se identifican los nodos de tipo *Variant* y se realiza una traslación de cada uno de ellos a *TextualRequirement*. La relación que une una variante con sus hermanos es una relación de exclusividad, por lo que accedemos al padre de la variante (*vp*) para acceder a la colección de requisitos hijos *col1*, de forma que con ellos podemos establecer una traza de exclusividad con los requisitos que generan.

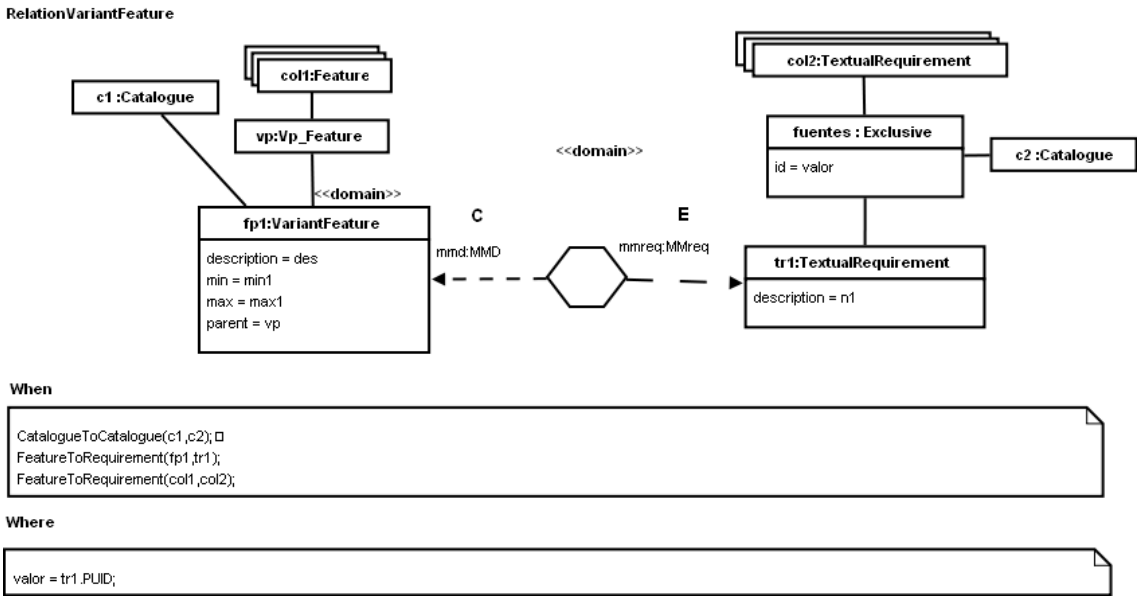

Figura 61 *RelationVariantFeature*

En la Figura 62 se realiza la misma correspondencia que en la Figura 61, pero aplicando el dominio de *matching* a una *feature* en vez de una *variantFeature*. Esto es así porque se permite que los puntos de variación se refinen en cualquier tipo de característica, incluyendo *vp_Feature*, lo que genera puntos de variación anidados. Por tanto en las relaciones de la Figura 61 y la Figura 62 se realizan los emparejamientos con la colección *refine_and-parent* y no con *variants-vpFeature*, con el fin de permitir mayor expresividad en el modelo.

Relaciones de composición

Las relaciones de composición entre características analizadas en el modelo MMD-FEAT.ecore son las siguientes: *conditions, favours, requires, excludes, refine_and* y *variants*. Las dos últimas ya se han repasado a lo largo de las transformaciones relativas a secciones y a puntos de variación. Toda relación de composición o construcción entre características proporciona una relación o traza en el dominio de MMreq.ecore. A cada característica que tenga una relación de composición se le ha asignado una relación de traza. Las relaciones de tipo *implements* y *parent-child* cumplen el mismo patrón, *node relationship*.

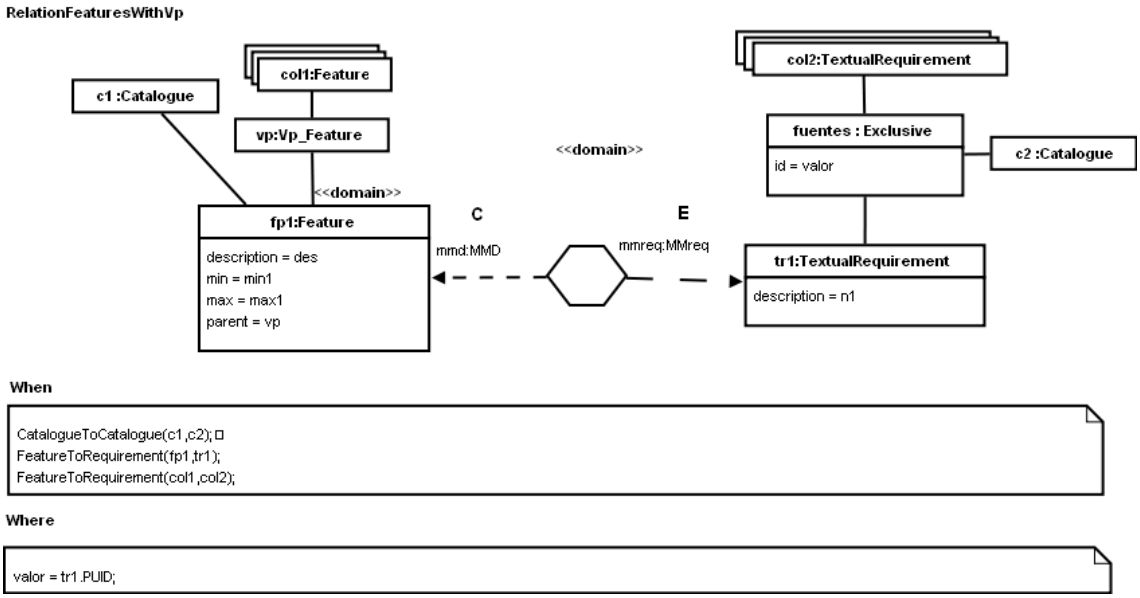

Figura 62 *RelationFeatureWithVp*

En la relación *RelationFeatureImplements* (Figura 63) se ilustra el ejemplo de relación de composición-traza. Ante una característica que está relacionada con un conjunto de características por la relación *implements*, se crea una relación de traza entre dicho requisito creador de la relación y su destino.

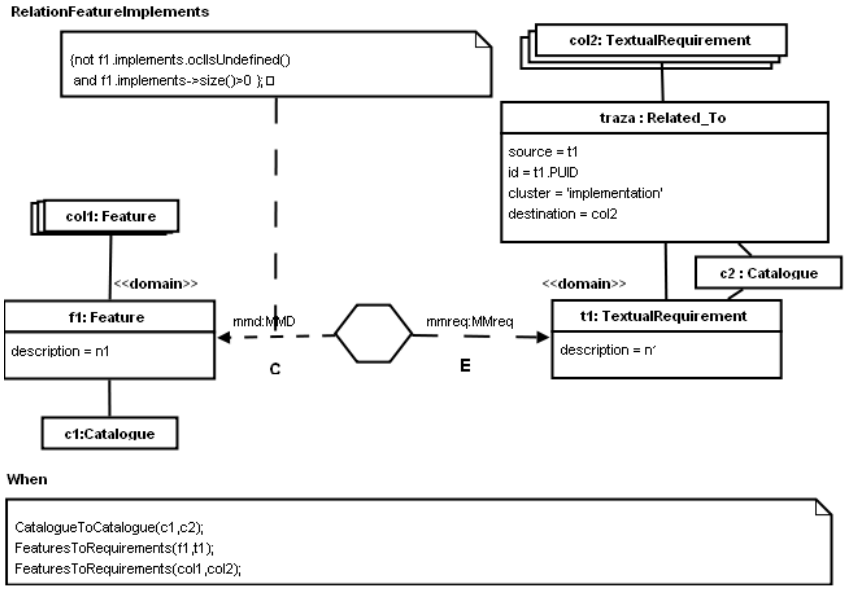

Figura 63 *RelationFeatureImplements*

Las siguientes reglas se nombran de forma análoga porque se atienen a un mismo patrón:

- *RelationFeatureImplements* equivale a traza *RelatedTo* cluster='implements'.
- *RelationFeatureConditions* equivale a traza *RelatedTo* cluster='conditions'.
- *RelationFeatureFavours* equivale a traza *RelatedTo* cluster='favours'.

- *RelationFeatureRequires* equivale a traza *Required.*
- *RelationFeatureExcludes* equivale a traza *Exclude.*
- *RelationFeatureParentChild* equivale a traza *ParentChild.*

A las trazas de estas relaciones se suman las trazas obtenidas de las relaciones relativas a secciones, puntos de variación y otros artefactos (trazas a otros artefactos, ver siguiente epígrafe).

Trazas a otros artefactos

La traza hacia artefactos que no son características (requisitos, plantillas de atributos de calidad, comandos) se debería plasmar en MMD-FEAT.ecore mediante relaciones *traceTo*. Como ejemplo de artefacto externo se muestra un *TextualRequirement* en el dominio de MMD-FEAT.ecore, que en el dominio destino debe mantener sus relaciones de traza.

En la relación *RelationTraceToFeature* (Figura 64) se muestra la correspondencia entre objetos *textualRequirement* de ambos dominios de inicio y destino junto con la relación *traceTo–traceFrom.* En la Figura 64 buscamos como patrón una característica con un conjunto de requisitos trazados, lo que implica la creación de cada uno de los requisitos en el dominio destino junto con una relación de traza *RelatedTo* que agrupa todos sus requisitos como destino.

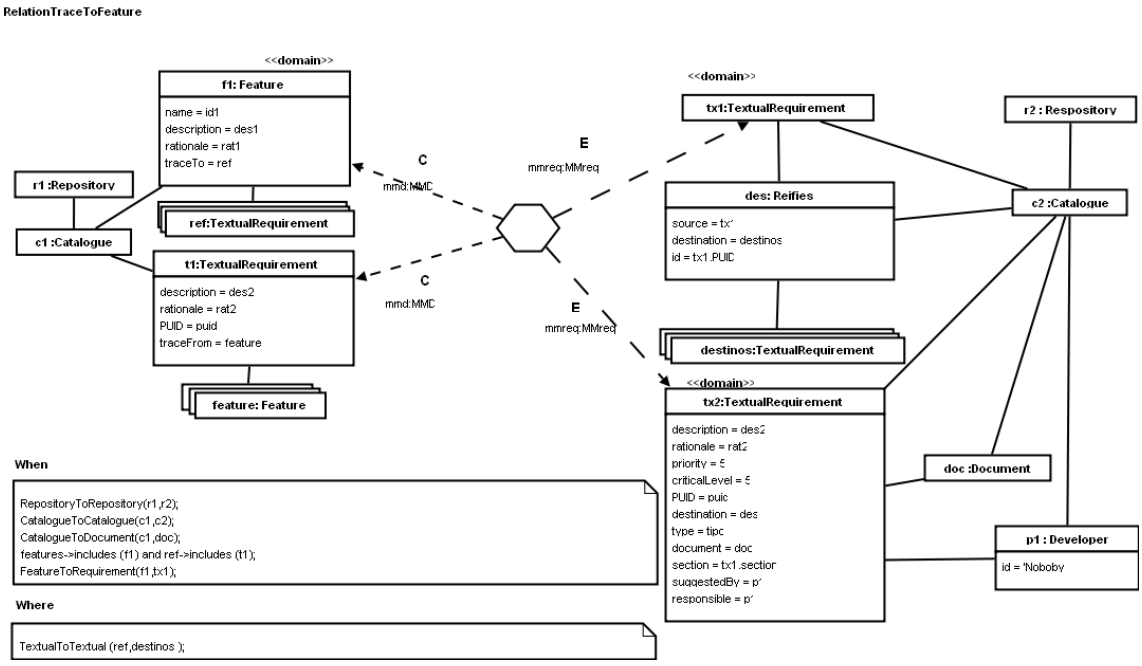

Figura 64 *RelationTraceToFeature*

En la relación *TextualToTextual* se realiza una sencilla correspondencia para la creación de los objetos instancia de *TextualRequirement*, de forma que ellos también puedan disparar esta regla.

4.4.8 Validación de las transformaciones

Los modelos desarrollados en el caso de estudio de los STO presentado en el Capítulo 3 se han usado para verificar la correcta implementación de las transformaciones y para validar la propuesta. Para la elaboración del archivo .xmi se ha construido un repositorio llamado *STO* y un catálogo del mismo nombre. Las 121 características utilizadas son alojadas en este catálogo, nombrándolas en anchura y por niveles de abstracción. Primero se listan en anchura las características de capacidad y después cada uno de los árboles de implementación del caso de estudio.

En la Figura 65 se muestra la vista que el editor *Ecore* proporciona del modelo STO.xmi elaborado conforme a MMD-FEAT.ecore. En la ventana de la izquierda se muestran las características del catálogo; en la ventana central están los requisitos generados y en la ventana de la derecha las propiedades del objeto seleccionado (*Feature EFTCOR* en este caso).

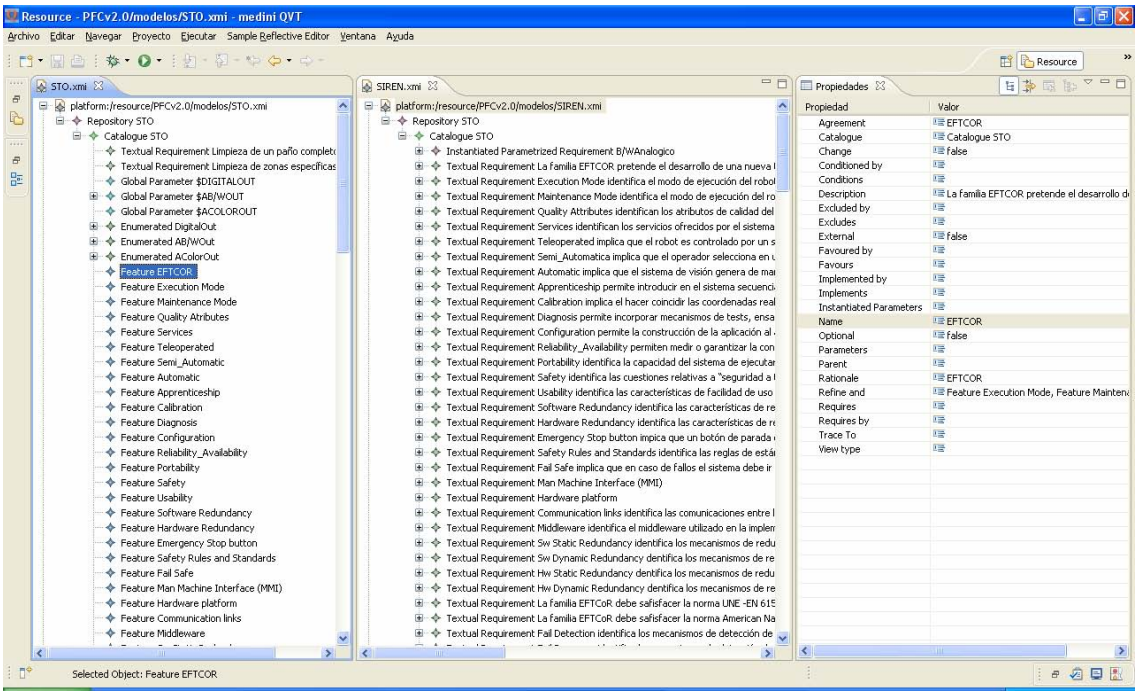

Figura 65 Vista IDE de STO.xmi y SIREN.xmi

La salida de las transformaciones es también un fichero de tipo .xmi, denominado *SIREN.xmi*, donde se encuentra un repositorio de catálogos con el mismo nombre que en la entrada de datos. En cada catálogo se tiene una lista de requisitos procedentes del modelo de características, obtenidos mediante la ejecución del *script Forward.qvt*. Las relaciones entre las características se mantienen a través de reglas de tipo *Trace*, hay una lista de los parámetros y tipos de datos referenciados, una colección de los interesados definidos y una instancia de documento organizado por secciones, cada una con una colección de referencias a los requisitos que alberga.

El tiempo de ejecución de los *scripts* es la suma de los tiempos de ejecución de cada regla, que viene determinado en función del número de objetos candidatos a la regla por dominio de *matching* (*checkonly*). En la Tabla 53 tenemos la información inicial de salida de ejecución del *script* sobre el número de objetos candidatos a cada regla. El

ejemplo elaborado *STO.xmi* tiene al menos 121 objetos *feature* candidatos, lo que nos debe producir al menos 121 requisitos. En la regla *RelationTraceToFeature* tenemos sólo introducidos dos *textualRequirement* como ejemplo * 121 *feature* = 242 tuplas (pares de objetos candidatos).

```
Took 0 ms totally for instantiate QVT engine
Took 32 ms totally for loading resource platform:/resource/PFCv2.0/modelos/STO.xmi
Took 15 ms totally for loading resource platform:/resource/PFCv2.0/modelos/SIREN.xmi
(Start QVT transformation)
QVT script Signature:[[Forward, [[mmd, [MMD]], [mmreq, [MMreq]]]]]
(QVT parsing and analysing done in 78 ms )
(QVT loading traces done in 63 ms )
(Relation 'RepositoryToRepository' initially has 1 tuple(s) to evaluate)
(Relation 'CatalogueToCatalogue' initially has 1 tuple(s) to evaluate)
(Relation 'CatalogueToDocument' initially has 1 tuple(s) to evaluate)
(Relation 'FeatureToRequirement' initially has 121 tuple(s) to evaluate)
(Relation 'FeatureToRootsection' initially has 121 tuple(s) to evaluate)
(Relation 'FeaturesToSections' initially has 121 tuple(s) to evaluate)
(Relation 'FeatureToSubsections' initially has 121 tuple(s) to evaluate)
(Relation 'FeaturesToRootSubsection' initially has 121 tuple(s) to evaluate)
(Relation 'RelationFeatureImplements' initially has 121 tuple(s) to evaluate)
(Relation 'RelationFeatureConditions' initially has 121 tuple(s) to evaluate)
(Relation 'RelationFeatureFavours' initially has 121 tuple(s) to evaluate)
(Relation 'RelationFeatureRequires' initially has 121 tuple(s) to evaluate)
(Relation 'RelationFeatureExcludes' initially has 121 tuple(s) to evaluate)
(Relation 'RelationFeatureParentChild' initially has 121 tuple(s) to evaluate)
(Relation 'FeatureExternalToRequirement' initially has 121 tuple(s) to evaluate)
(Relation 'FeatureChangeToRequirement' initially has 121 tuple(s) to evaluate)
(Relation 'Vp_FeatureToRequirement' initially has 18 tuple(s) to evaluate)
(Relation 'VariantFeatureToRequirement' initially has 22 tuple(s) to evaluate)
(Relation 'RelationVp_Feature' initially has 18 tuple(s) to evaluate)
(Relation 'RelationVariantFeature' initially has 22 tuple(s) to evaluate)
(Relation 'RelationFeaturesWithVp' initially has 121 tuple(s) to evaluate)
(Relation 'ParameterToParameter' initially has 3 tuple(s) to evaluate)
(Relation 'LocalParameterToLocalParameter' initially has 0 tuple(s) to evaluate)
(Relation 'GlobalParameterToGlobalParameter' initially has 3 tuple(s) to evaluate)
(Relation 'InstantiatedParameterToInstantiatedParameter' initially has 1 tuple(s) to evaluate)
(Relation 'FeatureWithParameterToParametrizedRequirement' initially has 121 tuple(s) to evaluate)
(Relation 'FeatureWithInstParameterToInstantiatedParametrizedRequirement' initially has 121
tuple(s) to evaluate)
(Relation 'RelationTraceToFeature' initially has 242 tuple(s) to evaluate)
(Relation 'DataTypeToDataType' initially has 10 tuple(s) to evaluate)
(Relation 'EnumeratedToEnumerated' initially has 3 tuple(s) to evaluate)
(Relation 'StringTypeToStringType' initially has 7 tuple(s) to evaluate)
(Relation 'StringTypeToStringType2' initially has 7 tuple(s) to evaluate)
(Relation 'NumberToNumber' initially has 0 tuple(s) to evaluate)
(Relation 'RangeToRange' initially has 0 tuple(s) to evaluate)
(Relation 'IntTypeToIntType' initially has 0 tuple(s) to evaluate)
(Relation 'FloatTypeToFloatType' initially has 0 tuple(s) to evaluate)
(Relation 'DataValueToDataValue' initially has 7 tuple(s) to evaluate)
(Relation 'StringValueToStringValue' initially has 7 tuple(s) to evaluate)
(Relation 'IntValueToIntValue' initially has 0 tuple(s) to evaluate)
(Relation 'FloatValueToFloatValue' initially has 0 tuple(s) to evaluate)
(start QVT evaluation in direction 'mmreq')
```

Tabla 53 Información inicial sobre la ejecución

El motor de QVT proporciona una salida sobre los resultados producidos que se muestra en la Tabla 54. En el ejemplo de los STO se indica el número de objetos creados, número de atributos creados o modificados y tiempo de finalización. En un ordenador de gama media actual el tiempo es inferior a 8 s. *(Plataforma: Intel Core 2 CPU T5500 a 1,66 GHz, 2048 MB RAM. Configuración: medini QVT Versión: 1.6.0.25263, jre 1.6.0_12-b04, Windows Xp Professional Service Pack 2.*)

(QVT evaluation done in 6984 ms, created 418 new elements, set 2406 features)
(QVT evaluateQVT() terminated after 7203 ms)
Allocate IDs
Begin storing traces
Saving traces to disk
Stopped saving traces to disk
(Storing traces done in in 516 ms)
(QVT transformation done in 7719 ms)

Tabla 54 Resultados producidos

La correspondencia de aplanamiento se ha probado directamente con los modelos resultado del caso de estudio de los STO, que fueron desarrollados en su momento sin el aplanamiento en mente. Al aplicar la generación se observa que la salida es heterogénea y de calidad variable, pues en ocasiones no se expresan con claridad las capacidades ni las decisiones de diseño. Por ejemplo, cuando el campo *Descripción* de las características, a partir del cual se genera un requisito textual directo, no indica explícitamente el nombre de la característica, o empieza con la cadena "Esta característica se refiere a (...)", el texto del requisito directo generado queda poco explicativo por sí solo, si no se concatena con el nombre del requisito generado, que se corresponde con la característica. Parece necesario, por tanto, proporcionar unas guías sencillas sobre cómo se debe escribir el texto de la descripción de las características de forma que la especificación de requisitos generada sea de mayor calidad. Estas guías consisten en un pequeño conjunto de patrones muy sencillos que no suponen ninguna limitación en la expresividad de la especificación de las características. Independientemente de que se recurra a un eventual proceso de aplanamiento, estos patrones pueden ayudar a escribir un modelo de características más homogéneo, ayudando a sistematizar la escritura de las características. Además, en relación con MMreq, nos hemos dado cuenta de que sería interesante disponer de un nombre para los requisitos, lo cual sería una extensión del modelo de referencia de requisitos de SIREN. En relación con el modelo de casos de uso, es preciso escribir el campo *Descripción* del caso de uso de forma que el requisito generado sea autocontenido y tenga sentido. Una propuesta inicial para estos patrones de escritura de la descripción de características es la siguiente:

- Para una característica *NombreCaracterísticaX* que reúne varias características hijas, se podría usar: "NombreCaracterísticaX *identifica* [listado de características hijas de NombreCaracterísticaX]."
- En el caso de las vp-features, un caso especial del anterior, tendríamos: "vp-featureX *identifica las distintas alternativas para* [descripción del objetivo del punto de variación definido por la vp-feature]."
- Las variantes se especificarían como requisitos hijos del requisito correspondiente a la vp-feature, de la siguiente manera:
    - $DOMMOD_{N1}$(dominio de la línea de productos): todas las variantes como hijos: "*La variante* X *implica* [descripción de la variante]." *o simplemente* "X *implica* [descripción de la variante]."
    - $DOMMOD_{N2}$ (producto instanciado): "*Se ha escogido la variante* X, *que implica* (...)."
- Para una característica hoja, que muestra una determinada alternativa de diseño o una capacidad concreta de la línea de productos, se puede usar:

"NombreCaracterísticaX *implica* [descripción de la capacidad o de la decisión de diseño]." o bien "NombreCaracterísticaX *permite* [descripción de la capacidad]."

- En el caso de que la característica se trate de una norma legal, estándar o reglamento, se podría usar (generalmente este tipo de cuestiones se refieren a la línea completa, no a un producto concreto): *"La línea de productos* NombreLínea *debe satisfacer* [CaracterísticaNorma]."

Estos sencillos patrones se han aplicado para reescribir el modelo de características del caso de estudio de los STO. Con esta segunda versión se ha observado un aumento de la calidad de la especificación generada con el aplanamiento y una mejora de la calidad de la especificación del propio modelo de características.

4.4.9 Sincronización

4.4.9.1 Motivación

Una cuestión importante cuando se pretende realizar una nueva generación del modelo destino después de un cambio en el modelo origen es qué hacemos con la información que ha sido añadida directamente en el modelo destino y que por tanto no está reflejada en el modelo origen. Si no realizamos una gestión adecuada de dicha información, ésta puede perderse al volver a realizar la generación. Los sistemas software de gran escala y complejidad se describen desde diferentes puntos de vista y niveles de abstracción, de manera que se reduce la complejidad de la especificación, pero a cambio se pueden producir inconsistencias cuando uno de los modelos es modificado. Situaciones como esta son habituales cuando modelos diferentes describen el mismo sistema desde puntos de vista distintos, pero no independientes, sino equivalentes en algún sentido. Por ejemplo, (1) cuando se describen los datos de un sistema mediante un modelo relacional de tablas y un modelo de clases UML; o (2) en el enfoque presentado en este capítulo, entre los conjuntos DOMMOD y REQSPEC, relacionados por la correspondencia de aplanamiento. Aparece, por tanto, un problema de sincronización entre modelos, ya que es necesario asegurar que dos modelos mantienen la consistencia.

Consideraremos resuelto nuestro problema de sincronización cuando se disponga de una herramienta que escale bien y que realice propagación de cambios en ambas direcciones preservando la nueva información escrita en ambos modelos, de forma que no se pierda el trabajo realizado al volver a generar. Al plantearnos el problema de la sincronización, pretendemos que si cualquier interesado (como cliente, usuario o analista) considera necesario modificar la definición de la línea de productos o de un producto concreto dentro de la línea, pueda realizar dicho cambio en la vista que prefiera (vista modelos o vista textual), para acto seguido propagar los cambios al modelo opuesto. Por ejemplo, en nuestro caso, con la sincronización se podría (1) añadir información al catálogo de requisitos generado en REQSPEC o realizar una modificación en los atributos de un requisito en REQSPEC, propagando después los cambios a DOMMOD; o bien (2) realizar cambios en DOMMOD, propagar tales cambios hacia adelante, y comprobar que los cambios que se habían realizado en REQSPEC no se han perdido. Nuestro objetivo es realizar una aproximación inicial a la problemática de la sincronización.

4.4.9.2 Antecedentes

Giese y Wagner [115] distinguen dos tipos de sincronización entre modelos: (1) la sincronización entre modelos por lotes o *batch*, donde se toma como entrada un modelo completo y se produce una salida usando un enfoque orientado a batería de comandos; y (2) la sincronización incremental entre modelos, en la que se propagan sólo los cambios realizados en el modelo origen, sin tener que rehacer totalmente el destino, de forma que el esfuerzo computacional se reduce.

Una solución al problema de la sincronización sería que la herramienta la gestionara automáticamente, como en la propuesta de Yingfei *et al.* [297], en la cual se proponen los retos principales de una herramienta que soporta sincronización automática entre modelos relacionados por un modelo de transformaciones: (1) para establecer y mantener la consistencia, se necesita establecer qué significa para estos dos modelos estar sincronizados; (2) se necesita una manera automática, derivada de la transformación, para realizar la propagación de cambios no sólo desde el modelo origen al modelo destino, sino también para reflejar los cambios desde el modelo destino al modelo origen: es decir, se necesitan *transformaciones bidireccionales*; y (3) el método de sincronización debería ser independiente de un modelo específico, o de un determinado lenguaje de transformación.

Como vemos una posible solución a la sincronización es el uso de transformaciones bidireccionales, sobre todo en problemas en los que el modelo fuente y el modelo destino pueden editarse independientemente. Una relación de transformación bidireccional (*bx*) es una relación que se mantiene entre ambos dominios en los dos sentidos. Stevens [277] expone las propiedades que debe poseer una transformación de modelos bidireccional expresada en QVT. Esta autora hace hincapié en que el hecho de que una transformación sea bidireccional no implica que sea biyectiva. Una transformación dada entre metamodelos *M* y *N* por la relación *R* es biyectiva si por cada modelo *m* conforme con *M* existe exactamente un modelo *n* conforme con *N* tal que *m* y *n* están relacionados por *R* y viceversa. Si *N* y *M* son metamodelos que se relacionan por un modelo de transformación, la relación $R \subseteq M \times N$ se mantiene entre un par de modelos –representada por *R (M,N)*– sii los modelos son consistentes entre sí. Asociadas a esta relación existen dos relaciones de transformación:

$$\overrightarrow{R} : M \times N \rightarrow N$$
$$\overleftarrow{R} : M \times N \rightarrow M$$

La idea es que $\overrightarrow{R}$ busque un par de modelos *(m,n)* y trabaje en cómo modificar *n* para asegurar la relación *R*. Por tanto $\overrightarrow{R}$ devuelve una versión modificada de *n*. De forma similar $\overleftarrow{R}$ propaga los cambios en la dirección opuesta. En una transformación QVT se asume que una transformación, independientemente de que sea biyectiva o no, debe ser determinista y debe producir el mismo resultado ante la misma entrada. También se asume que la función de transformación depende tanto del estado actual del modelo reemplazado, como del modelo origen, tal y como se hace en una relación QVT, ya que tanto el origen como el destino son entradas de la función y se utilizan en el emparejamiento de patrones.

4.4.9.3 Solución propuesta

En esta sección mostramos la solución técnica adoptada por Hernández [135] en el ámbito de su proyecto fin de carrera con el objetivo de dar un primer paso en el campo de la sincronización entre MMD.ecore y MMreq.ecore. La herramienta seleccionada para la implementación, mediniQVT [142], soporta transformación incremental entre modelos con propagación de cambios. A la hora de transformar o actualizar dos modelos, la herramienta debe recorrer primero el *script*, para conocer la estructura de las transformaciones, y después acude a su fichero de trazas (extensión *.traces*) para recuperar los objetos que han disparado reglas, de forma que si los objetos ya están creados no crea nuevos objetos, aunque sí propaga los cambios de los atributos de los objetos ya creados o crea nuevos objetos si estos no existían antes. Si realizamos cambios en el modelo origen (MMD) los cambios son propagados al modelo destino (MMreq) mediante el *script Forward.qvt*. Sin embargo, si se han realizado cambios en el modelo destino, estos cambios serían sobreescritos al reactualizar la información del modelo origen y volver a generar. Nuestra intención es evitar esta pérdida de información de tal forma que el analista o usuario pueda realizar modificaciones en el modelo destino y tales modificaciones no se pierdan al propagar cambios desde el modelo origen.

Se debe tener claro qué semántica tiene asociada una transformación en QVT-*relations*. Los dominios *checkonly* son dominios en modo sólo lectura que disparan reglas cuando se cumplen las precondiciones. Sin embargo, los dominios *enforce* se aseguran de que se cumpla lo expresado en el patrón aunque se requiera la modificación de los modelos objetivo. Un *script Forward.qvt* ha sido diseñado e implementado para la generación de requisitos textuales a partir de modelos de características y de casos de uso, por lo que las transformaciones son de la forma *checkonly*, sólo lectura en el modelo origen (MMD), y *enforce*, lectura-escritura en el modelo destino (MMreq). Cada vez que se ejecuta el *script* de actualización *Forward.qvt* el modelo origen se sincroniza con el modelo destino, de forma que el modelo destino refleja exactamente la información del modelo origen y por tanto los cambios que pudiera haber en el modelo destino se pierden. La solución para mantener esos cambios es reflejarlos hacia atrás antes de realizar una transformación hacia delante, por medio de un nuevo script *Sincronization.qvt*. Sólo se permite modificar uno de los modelos cada vez. Hay varias formas de detectar cuándo se ha de realizar sincronización: (1) implementar un programa que detecte cuando el usuario guarde el modelo destino e intente modificar el modelo origen, de tal forma que la herramienta le solicitaría realizar una sincronización antes; (2) bajo demanda del usuario, en cualquier momento; y (3) si la herramienta estuviera enfocada a un marco MVC (*Model-View-Controller*), podría forzarse la sincronización antes de que el controlador mostrara la vista.

El nuevo *script Sincronization.qvt* (ver Figura 66) es una transformación entre modelos bidireccional que cumple las propiedades propuestas por Stevens [277], de forma que mantiene la relación existente *R(n,m)* entre ambos modelos y en ambas direcciones. En realidad es un subconjunto de las transformaciones implementadas en *Forward.qvt*, de hecho reutiliza las trazas dejadas por el *script Forward.qvt* (es por ello que toma los mismos nombres de la transformaciones). Para lograr un *script* bidireccional se ha realizado un proceso de selección sobre las relaciones que no podían mantenerse en ambas direcciones (unidireccionales), tomando las siguientes directrices: (1) las transformaciones se han convertido en *enforce-enforce* pues ahora la transformación ha

de ser capaz de sincronizar en ambas direcciones. La dirección de la transformación (*mmd* o *mmreq*) indica qué dominio es el de lectura y cuál el de lectura-escritura. (Por ejemplo, al aplicar dirección *mmd* este modelo se convierte en el destino y por tanto lectura-escritura; sin embargo *mmreq* aun siendo de tipo *enforce* se convierte en sólo lectura); (2) deben eliminarse de las transformaciones los atributos no alcanzables y las asignaciones temporales (que en *Forward.qvt* se hacían constantemente para realizar el cálculo del *PUID* de cada requisito generado). Estas nuevas relaciones se denominan relaciones consistentes o *matching* estables; (3) deben eliminarse de la transformación bidireccional las relaciones unidireccionales (*FeatureToRootsection* o *FeatureToRootSubsection*) que tienen únicamente la intención de alcanzar estados intermedios; y (4) las transformaciones pueden crear nuevos elementos en ambos dominios, pero para ello deben cumplir las relaciones declaradas a fin de crearse adecuadamente.

Con este nuevo *script*, si algún interesado considera necesario modificar un requisito, sección o relación entre requisitos, puede realizarlo en el editor que le sea más cómodo (modelos gráficos o requisitos textuales) y propagar los cambios al modelo opuesto. Con la solución propuesta es posible [135] (1) añadir información al catálogo y realizar una modificación en los atributos de un objeto en el modelo destino (*SIREN.xmi*), para comprobar que los cambios se han propagado al modelo origen; y (2) realizar entonces cambios en el modelo origen, propagarlos hacia adelante, y comprobar que los cambios que se habían realizado en el modelo destino no se pierden. Se da así un primer paso en relación con uno de los objetivos que perseguimos en este Capítulo 4, tener dos vistas simultáneas, sincronizadas, del sistema o del software: la *vista modelos* y la *vista textual*. En una propuesta más madura sería necesario realizar un control de versiones que permitirá la *invertibilidad* (*undoability*) de las transformaciones como propone Stevens [277], de forma que en cualquier momento pudiera volverse a la versión anterior deshaciendo el cambio.

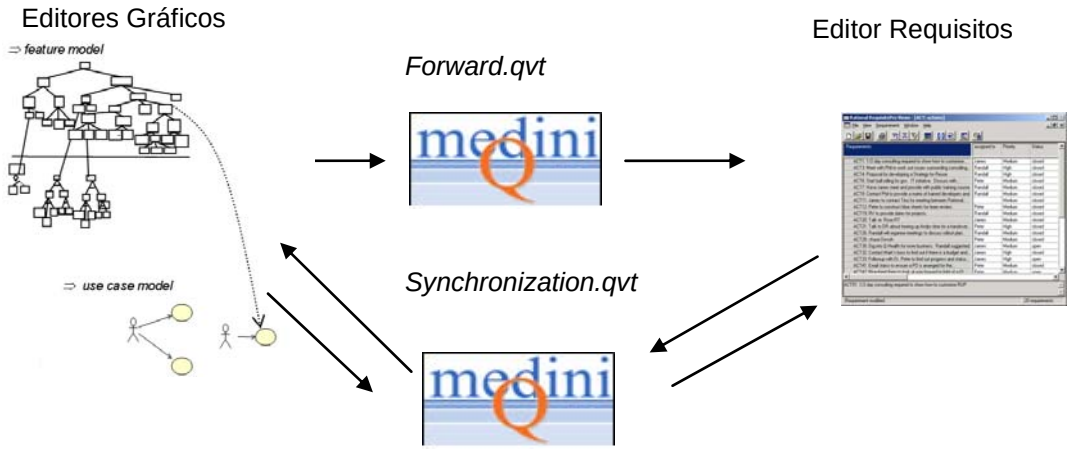

Figura 66 Marco de sincronización

4.5 Conclusiones

En este capítulo se ha presentado una propuesta para la generación de requisitos en lenguaje natural compatibles con el modelo de referencia de SIREN (Capítulo 2), a partir de los modelos de análisis del dominio propuestos en SIRENspl (Capítulo 3), lo

que denominamos una propuesta de *aplanamiento* de los modelos de SIRENspl en un modelo de requisitos en lenguaje natural.

La necesidad de caracterizar los dominios inicial y final de la correspondencia de aplanamiento, DOMMOD y REQSPEC, respectivamente, nos ha llevado a diseñar unos metamodelos que nos han permitido afinar la definición de las propuestas presentadas en capítulos anteriores de esta tesis doctoral, SIREN en el Capítulo 2 y SIRENspl en el Capítulo 3. Más concretamente, en el ámbito de DOMMOD, (1) el metamodelo de características MMD-FEAT (formado por 32 clases conceptuales) permite modelar diagramas de características complejos, puntos de variación anidados, parametrización, parametrización instanciada y el modelado de diferentes tipos de datos y relaciones entre características; (2) el metamodelo de casos de uso genéricos MMD-UC (formado por 20 clases conceptuales) permite modelar casos de uso genéricos junto con casos de cambio; y (3) el metamodelo de requisitos desarrollado para describir REQSPEC, MMreq (formado por 41 clases conceptuales), permite modelar requisitos SIREN 2.0 y un modelo abstracto de documento de requisitos que organiza los mismos por secciones.

Las transformaciones sencillas propuestas en la correspondencia de aplanamiento se han implementado en un entorno de desarrollo guiado por modelos, mediante el entorno proporcionado por Eclipse EMF y el lenguaje de transformaciones QVT.

4.5.1 La aportación al estado del arte

En el ámbito del análisis del dominio, nuestra propuesta constituye el primer enfoque que conocemos que proporciona dos *vistas* a los interesados, la *vista modelos* y la *vista textual*, con el objetivo de hacer más accesible la especificación del sistema a todos los interesados, de corte técnico y no técnico. Con esta propuesta las ideas de Davis sobre integración de modelos de ingeniería del software y requisitos en lenguaje natural se adaptan al ámbito de las líneas de productos, así como las ideas subyacentes al modelado literario de Arlow *et al.* [30] y la generación de requisitos textuales a partir de modelos de Meziane *et al*. [215] y de Maiden *et al*. [196]. La revisión sistemática que se ha realizado sobre la literatura de generación de requisitos textuales a partir de modelos de ingeniería del software es la primera que conocemos sobre este tema y nos parece que en sí misma ya es una aportación al estado del arte.

En términos de los campos utilizados en la síntesis de enfoques de generación de requisitos en lenguaje natural a partir de modelos (Tabla 47 y Tabla 49), nuestro enfoque se resume como se recoge en la Tabla 55. En dicha tabla el *Ámbito* se considera tanto *Requisito* como *Documento*, si bien el formato de documento que consideramos actualmente es un formato abstracto que todavía no se ha trasladado a un formato concreto (como IEEE 830). Nótese como en la columna *Validación*, el caso de estudio de los STO se ha etiquetado como CEA en lugar de CEI, lo cual es debido a que en los STO la aplicación de nuestro enfoque se ha realizado de forma retrospectiva, es decir, sobre los modelos generados durante el desarrollo del caso de estudio y no durante dicho desarrollo.

Llegados a este punto destacamos las siguientes aportaciones de nuestra propuesta:

- Ninguno de los enfoques de generación de requisitos en lenguaje natural a partir de modelos de ingeniería del software revisados en la revisión sistemática de la

literatura (Tabla 47 y Tabla 49) considera explícitamente la sincronización de los requisitos textuales generados, que caracterizan como mero texto generado. Nuestra propuesta, en cambio, se basa en la necesidad de sincronización entre los modelos de ingeniería del software y los requisitos textuales, en el marco de un modelo de procesos iterativo e incremental, con el objetivo de tener no sólo una instantánea textual de la especificación del sistema, sino dos vistas del sistema, la vista de modelos y la vista de texto.

- Se trata de la primera propuesta que conocemos que considera la generación de requisitos literarios en los modelos de características y de casos de uso genéricos.

Modo Comb.		**Ámbito**		**Modelo inicial**	**Texto generado**		**Método**	**Herramienta**	**Validación**
Gen.	Int.	Req.	Doc.		LF	LN			
■	□	■	■	Modelo de características; modelo de casos de uso genéricos	□	■	SIRENspl	Prototipo basado en Eclipse EMF y QVT	(CEA) STO
Leyenda: *Modo comb.* (Modo de combinación): Gen: Generador / Int: Integrador; *Ámbito*: Req: Requisito / Doc: Documento *Texto generado:* LF: Lenguaje Formal / LN: Lenguaje Natural; *Validación*: (CEA) Caso de Estudio Académico / (CEI) Caso de Estudio Industrial / (PI) Práctica Industrial; I-A: Investigación en Acción									

Tabla 55 Nuestro enfoque para SIRENspl y SIREN en términos de la síntesis de la Tabla 47 y Tabla 49

4.5.2 Conclusiones adicionales y vías futuras

Comenzamos esta sección planteando unas reflexiones sobre el proceso de aplicación de la correspondencia de aplanamiento de los modelos de análisis de SIRENspl en el caso de estudio de los STO. Evidentemente estas reflexiones se podrían enriquecer con la aplicación de la propuesta en varios casos de estudio reales, no de forma retrospectiva, como hemos hecho en esta tesis doctoral, sino durante el desarrollo mismo de una línea de productos y de los productos instanciados dentro de la línea. Reconocemos que una limitación de la validación presentada en la Sección 4.4 es que los resultados no se han validado en el contexto del equipo de desarrollo de los STO.

En relación con el aplanamiento del modelo de características de SIRENspl, decimos que generamos "requisitos textuales", pero en muchas ocasiones no son "requisitos" en sentido estricto: en la línea de productos, (1) con el aplanamiento en $DOMMOD_{N1}$ (el nivel de modelado del dominio), se obtiene una descripción textual de la arquitectura de alto nivel de la línea de productos, más que una especificación de requisitos del dominio propiamente dichos; y (2) con la generación en $DOMMOD_{N2}$ (el nivel del producto instanciado), se obtiene una documentación textual de las decisiones que se han adoptado en la instanciación de la línea de productos, más que una especificación de requisitos del producto propiamente dichos.

Lo expresado en el párrafo anterior se puede explicar con otras palabras, en el ámbito abordado en esta tesis, de la siguiente manera: en la línea de productos de los STO, un requisito podría ser, por ejemplo, *SYR1*. "El sistema EFTCoR deberá ser capaz de eliminar manchas aisladas a lo largo de todo el casco de un buque de entre X1 y X2 metros de eslora", donde *X1* y *X2* son parámetros. SYR1 es propiamente un requisito. Pero en el caso de estudio de los STO no nos quedamos en este nivel, sino que los requisitos generados dicen, por ejemplo: "El sistema EFTCoR tendrá un primario y opcionalmente un secundario", lo cual, más que un requisito, constituye una decisión de

diseño: el sistema EFTCoR se podría diseñar de otra manera para satisfacer SYR1. Todo esto procede del hecho de que, como comentamos en la Sección 3.3.2.1, con el modelo de características se abstrae una arquitectura de la línea de productos más que una especificación de requisitos *pura* (que no tenga en cuenta el diseño). Hecha esta puntualización, por supuesto que con el modelo de características también podemos especificar auténticos "requisitos" (principalmente de grano grueso), cuando hablamos de capacidades y características de calidad (estas últimas se corresponderían con requisitos no funcionales). En conclusión, los *ítems* generados con el aplanamiento se pueden considerar propiamente requisitos si consideramos que especifican (1) determinadas capacidades que tiene que tener la línea de productos o el producto final; o (2) restricciones de diseño, la manera en que necesariamente se deben implementar las capacidades de la línea de productos. Llamamos "requisitos textuales" a los elementos generados, aunque quizás son más bien "elementos de documentación de la arquitectura de la línea de productos o del producto instanciado". En realidad, el modelo generado en el proceso de aplanamiento no es más que una vista textual del modelo de características: como es lógico, no podemos abstraer de forma automática requisitos como el SYR1 anterior a partir del modelo de características: sólo podemos "transcribirlo", dando lugar a lo que denominamos requisitos directos y requisitos literarios.

Creemos que ambos tipos de requisitos, directos y literarios, expresados en lenguaje natural, son con carácter general más fáciles de comprender por todos los interesados que los modelos diagramáticos correspondientes, con lo cual se facilitaría la validación del modelo por parte de clientes y usuarios, se clarificaría la gestión del proyecto, se reduciría el tiempo para escribir una especificación de requisitos completa y se mejoraría la trazabilidad entre modelos y requisitos textuales.

En relación con el aplanamiento del modelo de casos de uso genéricos de SIRENspl, creemos que los resultados de la generación sí que se pueden denominar con propiedad "requisitos textuales". Ayuda a ello la plantilla de casos de uso que se propone en SIRENspl, que diferencia claramente entre las acciones del actor y las acciones del sistema. Por un lado, (1) las acciones del sistema son los requisitos propiamente dichos, siendo requisitos hijos del requisito generado en correspondencia con el caso de uso; y por otro lado (2) las acciones del actor (que estimulan el sistema) constituyen un contexto en el que se pueden ubicar los requisitos propiamente dichos (las acciones del sistema), y por eso en el aplanamiento se ponen entre paréntesis justo antes de los requisitos generados citados en (1).

Los 27 patrones literarios propuestos inicialmente en esta tesis doctoral son más sencillos que los presentados por Meziane *et al.* [215] en relación con los diagramas de clases UML o por Maiden *et al.* [196] en relación con los diagramas SD de i*. Los diagramas de características son más sencillos que las técnicas antes citadas y por ello esta conclusión no nos parece sorprendente.

Son múltiples las vías de trabajo futuro relacionadas con esta propuesta. Queda pendiente un estudio en mayor profundidad del papel de la especificación de requisitos textuales generados en un desarrollo iterativo e incremental de la línea de productos y de la instanciación de un producto en la línea de productos: por ejemplo cómo definir el ámbito de cada iteración en el documento de requisitos textuales generados, cuándo y cómo definir nuevas versiones. Estas cuestiones pueden ayudar a enriquecer el proceso

de IR del producto, que como afirman Käkölä y Dueñas [159] ha sido tradicionalmente descuidado en detrimento de la IR del dominio. En esta tesis doctoral se ha realizado una primera aproximación al problema de la sincronización entre la *vista modelos* y la *vista textual* de la línea de productos y del producto instanciado, pero una vía de trabajo futuro sería enriquecer esta propuesta inicial con el versionado de las especificaciones y de los modelos conforme el software va evolucionando. A través del proyecto fin de carrera de Lucía Cerdá [55], recientemente iniciado, se pretende desarrollar un entorno que integre la vista modelos desarrollada mediante SirenSPLTool (Sección 3.5) con una nueva vista textual que muestre los resultados de las transformaciones QVT presentadas en la Sección 4.4 y que considere la sincronización entre ambas vistas.

También como vía futura, se puede estudiar la ampliación del conjunto de técnicas *sincronizadas* con una vista textual, en particular cubriendo las técnicas de UML más relacionadas con los requisitos (modelo de clases, de casos de uso, de secuencia, de actividades), como un paso para que en la fábrica de software se disponga de una especie de *monitor de control del proyecto* con dos vistas, la vista modelos y la vista textual, obteniendo las ventajas de cada una de tales vistas en el nivel del análisis. Evidentemente este monitor de control podría abarcar múltiples cuestiones adicionales que no se abordan en esta tesis doctoral, como por ejemplo visualización de métricas, animación de las especificaciones mediante prototipos, sintonización y pruebas. La inclusión de las técnicas de SIRENspl que se han dejado fuera de esta propuesta de tesis doctoral, plantillas de atributos de calidad y comandos, probablemente no tienen un impacto importante en los resultados presentados pues se trata de técnicas muy ligadas al diseño más que a la especificación de requisitos.

En relación con los metamodelos que describen DOMMOD y REQSPEC, (1) una posible vía futura de extensión de MMD-FEAT sería la extensión de la variabilidad como hace el modelo de variabilidad ortogonal de Pohl *et al.* [249]; (2) una posible extensión de MMD-UC sería la posibilidad de modelar los escenarios de interacción de los subsistemas del sistema (actualmente el sistema se modela sólo como una caja negra); (3) la definición de documento de requisitos que se encuentra en REQSPEC se puede extender en el sentido de la propuesta de John [154], que presenta un metamodelo de documentos de requisitos. Se podría estudiar también la correspondencia entre este formato abstracto de documento de requisitos, definido en REQSPEC, y formatos concretos de documento de requisitos, como el estándar IEEE 830 [138] o la propuesta de Volere [265]. Se trataría de desarrollar adaptadores que sirvieran para traducir el formato de documento abstracto REQSPEC a formatos concretos, incluso definidos por el usuario. Se trata de una función que ya ofrecen herramientas CARE del mercado.

La vista textual podría dar lugar a *instantáneas de la especificación de requisitos*, en forma de documentos de requisitos, que podrían definirse a varios niveles, por ejemplo: (1) para el cliente (por ejemplo, una versión completa que sirva como contrato o para validación; una versión “de alto nivel”, distinta de la anterior, para transmitir rápidamente las funcionalidades de la línea de productos, por ejemplo, de cara a su comercialización); (2) para los usuarios (con vistas a la validación); (3) para los ingenieros de pruebas (con el objetivo de obtener listas de chequeo en las pruebas del software, del sistema o de validación); y (4) para los diseñadores (incluyendo los requisitos funcionales del sistema, los requisitos no funcionales y las restricciones de diseño). Incluso, yendo más allá, se podría estudiar la traducción automática a distintos idiomas de los patrones de generación y del cuerpo mismo de los requisitos. El

vocabulario definido en el esquema conceptual del dominio podría ser de utilidad aquí para *normalizar* los resultados de la traducción, de manera que no se alejara del vocabulario utilizado por los expertos en el dominio. Siguiendo esta idea, la documentación estaría disponible en distintos idiomas. En el contexto de un desarrollo global de software, esta funcionalidad sería extremadamente interesante: independientemente de que se haya definido un idioma de trabajo común para el proyecto, muchos interesados agradecerían tener la especificación en su lengua materna.

5 Conclusiones y vías futuras

5.1 Análisis de la consecución de objetivos

En las secciones de conclusiones de los capítulos 2, 3 y 4 se han recogido las reflexiones detalladas en relación con cada una de las tres líneas de investigación en Ingeniería de Requisitos (IR) que confluyen en esta tesis doctoral, junto con una descripción de posibles vías de trabajo futuro. En esta sección resumimos las principales aportaciones de la investigación llevada a cabo en el marco de esta tesis doctoral, para mostrar cómo tales aportaciones desarrollan los objetivos planteados en la Sección 1.2, "Hipótesis y Objetivos".

Objetivo 1. Estudiar la reutilización de requisitos textuales, definiendo un método de IR basado en reutilización que incluya un metamodelo de requisitos, un conjunto de técnicas, un modelo de procesos y una herramienta de soporte.

- Después de una breve descripción del estado del arte en IR y en reutilización de requisitos, en el Capítulo 2 de esta memoria se ha descrito en profundidad SIREN, un método de IR basado en reutilización que trata con requisitos escritos preferentemente en lenguaje natural. Esta descripción de SIREN incluye los elementos mencionados en el Objetivo 1: (1) el metamodelo de requisitos o modelo de referencia de requisitos de SIREN, que primero se describe informalmente en el Capítulo 2 y después se formaliza en el Capítulo 4, utilizando MOF y Ecore; (2) el conjunto de técnicas de SIREN, que incluye guías para la definición de requisitos, atributos de los requisitos, requisitos parametrizados, trazas, organización de los catálogos, reutilización de requisitos, mejora de los catálogos y organización de los documentos de requisitos; (3) el modelo de procesos de SIREN, con una descripción de sus tareas y subtareas; y en último lugar (4) la herramienta de soporte de SIREN, SirenTool. Finalmente, en el Capítulo 2 se ha mostrado SIRENgsd, una propuesta de extensión de SIREN para el desarrollo global de software, junto con un repositorio de amenazas y salvaguardas para la IR que se desarrolla en un entorno globalizado, repositorio que se basa en una revisión sistemática de la literatura.

Objetivo 2. Definir un metamodelo de variabilidad en una línea de productos software, que sea compatible con los resultados del Objetivo 1, y establecer un conjunto de técnicas que representen dicho modelo de variabilidad en el nivel de los requisitos.

- En el Capítulo 3 de esta memoria se ha descrito SIRENspl, una evolución de SIREN con el objetivo de soportar la IR para líneas de producto en el contexto de un problema concreto, el dominio de los sistemas teleoperados para mantenimiento de cascos de buques (STO). SIRENspl se basa en técnicas ya existentes en análisis del dominio, que se han seleccionado y particularizado después de un estudio del estado del arte en IR para líneas de productos, con el objetivo de modelar adecuadamente el caso de estudio de los STO. En el Capítulo 3 el metamodelo de variabilidad de la línea de productos se describe sólo implícitamente a través de la presentación informal de las técnicas involucradas en SIRENspl; realmente es en el Capítulo 4 cuando se formaliza

dicho metamodelo –los modelos de características y de casos de uso genéricos– utilizando MOF y Ecore. Por tanto el proceso seguido en la tesis doctoral ha seguido este orden (el inverso al enunciado en el Objetivo 2): primero se han seleccionado unas técnicas para el análisis del dominio, que incluyen la definición de la variabilidad de la línea de productos, y después se han formalizado.

Objetivo 3. Definir un proceso de generación de documentación de requisitos textuales, en un proceso de desarrollo de software iterativo e incremental, que sincronice los modelos definidos en el Objetivo 2 con el metamodelo de requisitos textuales establecido en el Objetivo 1.

- En el Capítulo 4 de esta memoria se ha comenzado planteando intuitivamente el interés de la integración de modelos de ingeniería del software con requisitos textuales en lenguaje natural mediante la generación de requisitos textuales candidatos a partir de modelos de ingeniería del software (preferentemente modelos gráficos). Una vez planteado intuitivamente el interés de esta línea de investigación, se ha realizado una revisión sistemática de la literatura sobre generación de requisitos textuales a partir de modelos, que ha corroborado el interés de esta línea de trabajo. Acto seguido, se ha propuesto lo que denominamos una correspondencia de aplanamiento de los modelos de análisis del dominio propuestos en SIRENspl en requisitos en lenguaje natural compatibles con SIREN. Para formalizar esta propuesta se ha modelado formalmente con MOF los dominios de inicio y destino de la correspondencia de aplanamiento, esto es (1) como dominio de inicio, las técnicas de SIRENspl –acotando a modelos de características y casos de uso genéricos–; y (2) como dominio de destino, el modelo de referencia de requisitos de SIREN. El deseo de utilizar esta correspondencia de aplanamiento en un desarrollo iterativo e incremental nos ha llevado a plantear el problema de la sincronización de los modelos de inicio y destino.

Objetivo 4. Diseñar e implementar el prototipo de una herramienta que soporte los modelos definidos en 2 y el proceso de generación definido en 3.

- En el Capítulo 3 de esta memoria se ha descrito un prototipo básico de herramienta de soporte a SIRENspl, SirenSPLTool, desarrollado mediante Eclipse EMF/GMF, con el cual se da soporte a los modelos de análisis del dominio definidos en SIRENspl a raíz del Objetivo 2. Posteriormente, en el Capítulo 4 se ha mostrado una implementación de la correspondencia de aplanamiento propuesta, utilizando para ello Eclipse EMF junto con un lenguaje declarativo de transformación de modelos, QVT-relations. Se proporciona también una primera aproximación al problema de la sincronización de los modelos de inicio y destino.

Objetivo 5. Validar los resultados anteriores en al menos un caso de estudio real.

- En relación con el Objetivo 1, SIREN ha sido validado en entornos industriales fundamentalmente a través del proyecto GARTIC (Sección 2.7.3). En relación con el Objetivo 2, el enfoque propuesto por SIRENspl ha sido validado con su aplicación mediante Investigación-Acción en el caso de estudio de los STO,

como se muestra en el Capítulo 3. Finalmente, en relación con los objetivos 3 y 4, la herramienta SirenSPLTool y la implementación de la correspondencia de aplanamiento con QVT-relations han sido validadas con su aplicación retrospectiva a los modelos generados en el caso de estudio de los STO.

La hipótesis de partida de esta tesis doctoral decía que es factible definir una propuesta de IR para líneas de productos que facilite la reutilización de requisitos y que integre modelos de ingeniería del software y requisitos textuales (Sección 1.2). Creemos que el desarrollo de los objetivos anteriores permite probar dicha hipótesis de partida y satisface el objetivo global de esta tesis doctoral: "Definir una propuesta de gestión de requisitos para líneas de productos que integre modelos de ingeniería del software y requisitos textuales en lenguaje natural."

5.2 Contraste de resultados de la investigación

A lo largo del desarrollo de esta tesis doctoral se han publicado diversas contribuciones en diferentes foros científicos. En esta sección se compilan estos resultados de la investigación. Para ello, en primer lugar, en la Tabla 56 se muestra una clasificación de las publicaciones y trabajos realizados. El 6+3 que figura en la celda correspondiente a los proyectos fin de carrera simboliza seis proyectos ya leídos más tres proyectos actualmente en curso.

Tipo de contribución	**Total**
Artículos en revistas internacionales ISI/JCR	4
Capítulos de libro	4
Congresos, talleres y reuniones científicas internacionales	2
Congresos, talleres y reuniones científicas nacionales	8
Proyectos fin de carrera	6+3
Informes técnicos	1
Total de contribuciones	**28**

Tabla 56 Número de publicaciones y trabajos relacionados con la tesis, por tipos

En las siguientes secciones se desglosan las contribuciones anteriores, por categorías.

5.2.1 Artículos en revistas internacionales ISI/JCR

En la Tabla 57 se muestran los artículos en revistas que figuran en índices de impacto ISI/JCR (*Journal Citation Reports*). Obsérvese como cada una de las tres líneas de investigación que confluyen en esta tesis doctoral (detalladas en los capítulos 2, 3 y 4) está respaldada por al menos un artículo de este tipo: (1) reutilización de requisitos en lenguaje natural, por [284] y [280]; (2) reutilización de requisitos en el ámbito de la línea de productos que conforman los sistemas teleoperados para mantenimiento de cascos de buques, por [225]; y (3) generación de requisitos en lenguaje natural a partir de modelos de ingeniería del software, por [229].

Referencia	Publicación
(Toval *et al.* 02) [284]	Ambrosio Toval, Joaquín Nicolás, Begoña Moros, Fernando García (2002). *Requirements Reuse for Improving Information Systems Security: A Practitioner's Approach.* **Requirements Engineering Journal**. Vol. 6 (1): 205-219. Factor de impacto = 1,625 (2008 JCR).
(Toval *et al.* 08) [280]	Ambrosio Toval, Begoña Moros, Joaquín Nicolás, Joaquín Lasheras (2008). *Eight Key Issues for an Effective Reuse-Based Requirements Process.* **International Journal of Computer Systems Science and Engineering**. Vol. 23(6). Factor de impacto = 0,277 (2008 JCR).
(Nicolás *et al.* 09a) [225]	Joaquín Nicolás, Joaquín Lasheras, Ambrosio Toval, Francisco J. Ortiz, Bárbara Álvarez (2009). *An Integrated Domain Analysis Approach for Teleoperated Systems*. **Requirements Engineering Journal**. Vol. 14 (1): 27-46. Factor de impacto = 1,625 (2008 JCR).
(Nicolás y Toval 09) [229]	Joaquín Nicolás, Ambrosio Toval (2009) *On the Generation of Requirements Specifications from Software Engineering Models: A Systematic Literature Review.* **Information & Software Technology**. Vol. 51 (9): 1291-1307. Factor de impacto = 1,200 (2008 JCR).

Tabla 57 Artículos en revistas en índices de impacto ISI/JCR

5.2.2 Capítulos de libro

En la Tabla 58 se muestran los capítulos de libro publicados. Nótese como el capítulo (Nicolás y Toval 09) es la segunda edición, revisada y ampliada, del capítulo (Nicolás y Toval 07).

Referencia	Publicación
(Toval *et al.* 01) [281]	Ambrosio Toval, Joaquín Nicolás, Begoña Moros (2001): Requisitos Reutilizables de Seguridad en Sistemas de Información y Bases de Datos. En *Seguridad en Bases de Datos*, Ed. Dintel: Madrid, pp. 269-300. ISBN: 84-931933-9-9.
(Toval *et al.* 02) [283]	Ambrosio Toval, Joaquín Nicolás, Begoña Moros (2002): SIREN: Un Proceso de Ingeniería de Requisitos Basado en Reutilización. En *Applying Requirements Engineering,* Ed. Catedral Publicaciones: Sevilla, pp. 57-72. ISBN 84-96086-06-2.
(Nicolás y Toval 07) [227]	Joaquín Nicolás, Ambrosio Toval (2007): Gestión de Requisitos. En *Fábricas de Software: Experiencias, Tecnologías y Organización,* Ed. RA-MA: Madrid, pp. 143-176. ISBN 978-84-7897-809-0.
(Nicolás y Toval 09) [228]	Joaquín Nicolás, Ambrosio Toval (2009): Gestión de Requisitos. En *Fábricas de Software: Experiencias, Tecnologías y Organización (2ª edición),* Ed. RA-MA: Madrid, 32 pp. *Pendiente de publicación. Publicación estimada otoño de 2009.*

Tabla 58 Capítulos de libro

5.2.3 Congresos, talleres y reuniones científicas internacionales

Referencia	Publicación
(Nicolás *et al.* 06) [224]	Joaquín Nicolás, Joaquín Lasheras, Ambrosio Toval, Francisco J. Ortiz, Bárbara Álvarez (2006): A Collaborative Learning Experience in Modelling the Requirements of Teleoperated Systems for Ship Hull Maintenance, *Learning Software Organizations + Requirements Engineering (LSO+RE 2006)*, Hannover (Alemania), 27 y 28 de marzo de 2006.
(López *et al.* 09) [187]	Alejandro López, Joaquín Nicolás, Ambrosio Toval (2009): A Repository of Risks and Safeguards for the Requirements Engineering Process in Global Software Development. KNOWING'09, Limerick (Irlanda), 13 de julio de 2009.

Tabla 59 Congresos, talleres y reuniones científicas internacionales

5.2.4 Congresos, talleres y reuniones científicas nacionales

Referencia	Publicación
(Toval *et al.* 01) [282]	Ambrosio Toval, Joaquín Nicolás, Begoña Moros (2001): SIREN: Un Proceso de Ingeniería de Requisitos Basado en Reutilización. *Jornadas de Ingeniería de Requisitos Aplicada*, Sevilla, 11-12 de junio de 2001.
(Lasheras *et al.* 03) [177]	Joaquín Lasheras, Ambrosio Toval, Joaquín Nicolás, Begoña Moros (2003): Soporte automatizado a la reutilización de requisitos. *VIII Jornadas de Ingeniería de Software y Bases de Datos*, Alicante, 12-14 de noviembre de 2003.
(Lasheras *et al.* 04a) [178]	Joaquín Lasheras, Ambrosio Toval, Joaquín Nicolás, Begoña Moros (2004): Definición de Requisitos de Seguridad con Fines de Reutilización, *Primer Taller de Seguridad en Ingeniería del Software y Bases de Datos*, Málaga, 9 de noviembre de 2004.
(Lasheras *et al.* 04b) [176]	Joaquín Lasheras, Joaquín Nicolás, Ambrosio Toval, Begoña Moros (2004): Hacia un Modelo del Dominio de los Sistemas Teleoperados a través de una Extensión de SIREN, *II Jornadas de Trabajo DYNAMICA*, Málaga, 11 de noviembre de 2004.
(Martínez *et al.* 05a) [206]	Miguel A. Martínez, Joaquín Lasheras, Joaquín Nicolás, Ambrosio Toval (2005): Aplicación de un proceso de auditoría de datos personales basado en el método SIREN, *III Jornadas de Trabajo DYNAMICA*, Almagro (Ciudad Real), 21 y 22 de abril de 2005.
(Nicolás *et al.* 05a) [222]	Joaquín Nicolás, Joaquín Lasheras, Ambrosio Toval, Begoña Moros, Pedro Sánchez, Bárbara Álvarez (2005): Ingeniería de requisitos basada en reutilización: una propuesta de aplicación a los sistemas teleoperados para limpieza de cascos de buques, *III Jornadas de Trabajo DYNAMICA*, Almagro (Ciudad Real), 21 y 22 de abril de 2005.
(Martínez *et al.* 05b) [207]	Miguel A. Martínez, Joaquín Lasheras, Joaquín Nicolás, Ambrosio Toval (2005): Un proceso de auditoría de datos personales basado en Ingeniería de Requisitos, I Simposio sobre Seguridad Informática (incluido en CEDI 2005), Granada, 14 a 16 de septiembre de 2005.
(Nicolás *et al.* 05b) [223]	Joaquín Nicolás, Joaquín Lasheras, Ambrosio Toval, Francisco J. Ortiz, Bárbara Álvarez (2005): Una experiencia de modelado de los sistemas teleoperados para limpieza de cascos de buques mediante características y casos de uso genéricos, *IV Jornadas de Trabajo DYNAMICA*, Archena (Murcia), 17 y 18 de noviembre de 2005.

Tabla 60 Congresos, talleres y reuniones científicas nacionales

5.2.5 Proyectos fin de carrera

En la Tabla 61 se recogen los proyectos fin de carrera dirigidos en el marco de esta tesis doctoral. El asterisco en las tres últimas filas indica que los proyectos correspondientes están todavía en curso.

Referencia	Publicación
(Lasheras 03) [175]	Joaquín Lasheras Velasco (2003): *Análisis y prototipado de una herramienta CARE de soporte al método SIREN: SirenTool*. Proyecto Informático. Directores: Ambrosio Toval, Joaquín Nicolás y Begoña Moros. Convocatoria: marzo de 2003. Calificación: Sobresaliente
(Martínez 05) [205]	Miguel Ángel Martínez Aguilar (2005). *Revisión de los perfiles SIREN de Protección de Datos Personales y de Seguridad y aplicación a un caso de estudio real*. Directores: Ambrosio Toval, Joaquín Nicolás y Begoña Moros. Convocatoria: junio de 2005. Calificación: Sobresaliente.
(López 08) [185]	Antonio Alejandro López Lorca (2008). *SIRENgsd: Una Propuesta para la Reutilización de Requisitos en el Desarrollo Global de Software*. Director: Joaquín Nicolás. Convocatoria: septiembre de 2008. Calificación: Sobresaliente
(Hernández 09) [135]	Jorge Hernández Pérez (2009). *Generación y Sincronización de Requisitos Textuales desde Modelos de Características*. Directores: Joaquín Nicolás y Ambrosio Toval Convocatoria: junio de 2009. Calificación: Sobresaliente.
(Pla 09) [248]	Ana Pla Micó (2009). *Generación de Requisitos Textuales Incluyendo Literarios desde Modelos de Características y de Casos de Uso Genéricos*. Director: Joaquín Nicolás. Convocatoria: junio de 2009. Calificación: Sobresaliente.
(Caro 09) [53]	Gustavo Caro Rosillo (2009). *Herramienta CARE para el Modelado de Requisitos de Líneas de Producto*. Director: Joaquín Nicolás. Convocatoria: septiembre de 2009. Calificación: Sobresaliente.
(García 10)* [114]	María García Martínez (2010). *Una Propuesta para la Actividad de Negociación de Requisitos en el Desarrollo Global de Software*. Director: Joaquín Nicolás. Convocatoria: febrero de 2010 (estimada).
(Madrigal 10)* [190]	Mariano Madrigal Arques (2010). *Herramienta CARE de Soporte a SIRENgsd*. Director: Joaquín Nicolás. Convocatoria: febrero de 2010 (estimada).
(Cerdá 10)* [55]	Lucía Cerdá López (2010). *Herramienta CARE para la Integración de Requisitos Textuales y Modelos en Ambientes de Líneas de Productos*. Director: Joaquín Nicolás. Convocatoria: febrero de 2010 (estimada).

Tabla 61 Proyectos fin de carrera de la Facultad de Informática de la Universidad de Murcia

5.2.6 Informes técnicos

Referencia	Publicación
(Nicolás *et al.* 09b) [226]	Joaquín Nicolás, Begoña Moros, Joaquín Lasheras, Ambrosio Toval, *SIREN: Un Método Práctico de Ingeniería de Requisitos Basado en Reutilización, Versión 2.0*. 2009, DIS tech. report, TR DIS 1-2009, Departamento de Informática y Sistemas, Universidad de Murcia.

Tabla 62 Informes técnicos

5.3 Líneas de trabajo futuro

Aunque creemos que las tres líneas de investigación desarrolladas en el marco de esta tesis doctoral suman de forma natural y confluyen en el desarrollo del objetivo global de la tesis y en la consecuente validación de la hipótesis de partida, pensamos que la investigación es atípica en cierto sentido, dado que se dirige a numerosos problemas de naturaleza distinta, aunque complementaria. El trabajo realizado a lo largo de esta tesis doctoral se puede continuar por tanto de múltiples maneras; es más, nos parece que se puede proseguir por tantos caminos diferentes que consideramos necesario seleccionar unos en detrimento de otros y acotar así las líneas de trabajo futuro, de manera que éste se realice menos en anchura y más en profundidad. En las secciones de conclusiones de

los capítulos 2, 3 y 4 se puede encontrar una descripción detallada de posibles vías de trabajo futuro relacionado con cada una de las líneas de trabajo. En este apartado mostraremos una visión general de las líneas de trabajo futuro de este doctorando.

Siempre en el marco de la IR, las líneas de investigación que este doctorando pretende abordar desde un futuro inmediato son (1) la IR para desarrollo de software global, a través del proyecto PANGEA (ver Tabla 8 en la página 44), continuando el trabajo descrito en la Sección 2.10; y (2) la integración de modelos de ingeniería del software con especificaciones de requisitos en lenguaje natural. Ambas líneas de trabajo no son necesariamente disjuntas, dado que la segunda línea se puede estudiar en el contexto específico del desarrollo global de software.

- En el desarrollo de la primera línea, la IR para desarrollo de software global, son muchas las cuestiones que permanecen abiertas. Actualmente se ha iniciado el estudio de la negociación globalizada de requisitos, con el objetivo de dotar a SIRENgsd de una actividad de negociación de requisitos más detallada. También se ha comenzado la implementación de un prototipo de una herramienta CARE de soporte a SIRENgsd. En el corto plazo es imperativo plantear además la validación de todas estas propuestas, lo que puede llevar a mejorar SIRENgsd y el repositorio de buenas prácticas (salvaguardas) ante las amenazas que plantea el desarrollo global de software.
- En el desarrollo de la segunda línea, la integración de modelos de ingeniería del software con especificaciones de requisitos en lenguaje natural, son también muchas las cuestiones que permanecen abiertas. Se puede ampliar el conjunto de modelos desde los cuales se ofrezca la posibilidad de generar requisitos, incluyendo los modelos de UML más directamente ligados con la especificación de requisitos (como modelos de casos de uso, modelos de actividades, modelos de secuencia y modelos de clases). También se puede completar la generación de requisitos literarios y de documentos de requisitos con el objetivo de obtener una vista de la especificación –y por tanto del sistema en desarrollo o ya desarrollado– que sea más asequible a todos los interesados (*stakeholders*). Se debe profundizar el estudio de la sincronización de la *vista modelos* y de la *vista textual*, que en un ciclo de desarrollo iterativo e incremental permita registrar cambios en la especificación en forma más legible para todos los interesados.

Finalmente, un último comentario que sirve a la vez de conclusión y de forma de enfocar el trabajo futuro. Como hemos comentado en esta memoria de tesis doctoral, la propuesta de SIRENgsd no ha sido validada en un caso de estudio real y la propuesta de aplanamiento no ha sido validada más que con su aplicación retrospectiva a los modelos resultado del caso de estudio de los STO. A este respecto queremos realizar un comentario especial sobre la validación de la investigación en IR. La experiencia acumulada durante estos años nos hace conscientes de la dificultad de la validación empírica de las propuestas que se realizan en ingeniería del software en general y en IR en particular. Según nuestra experiencia es difícil encontrar empresas y organizaciones que estén dispuestas a aplicar los resultados de la investigación procedentes de un grupo de investigación universitario. A lo largo de esta tesis nos hemos encontrado con la resistencia al cambio de varios responsables. El proyecto GARTIC, con cinco empresas participantes, puede servir como contraejemplo de lo que acabamos de comentar: en los últimos tiempos parece que se observa un cambio, motivado tal vez en parte por la inquietud de las empresas ante los procesos de certificación. En ocasiones los

profesionales, que se encuentran con problemas que deben resolver a diario sin las herramientas conceptuales adecuadas, son más proclives a la innovación que los responsables organizativos, pero sin el soporte de estos últimos la innovación no es posible. Por otro lado, aún cuando se encuentra un entorno favorable para la experimentación, como el proyecto GARTIC anteriormente citado, es difícil probar las ganancias de productividad y calidad que se obtienen con las nuevas propuestas, si las hubiera, pues en muchas empresas y organizaciones no existen registros previos de actividad. En general nos encontramos con experiencias a partir de las cuales es difícil generalizar conclusiones y por supuesto no se puede garantizar la repetibilidad. Además, en ocasiones los profesionales modifican a la carta las propuestas que se pretenden validar, adaptándolas a su cultura o al contexto de aplicación, pero comprometiendo de nuevo la repetibilidad. Todo esto nos lleva a una conclusión sobre la forma de abordar el trabajo futuro, una conclusión que a miembros de grupos de investigación más maduros parecerá evidente: siguiendo la terminología empleada en el Capítulo 1, buscar más el desarrollo de casos de estudio que de experiencias; pensar en la validación ya en el inicio de la investigación; abordar problemas diseñando desde el principio la estrategia que se va a aplicar en la validación; sólo abordar problemas cuya validación sea factible. Si estas cuestiones no se tienen en cuenta se pueden formular propuestas que podemos considerar interesantes y bien fundamentadas, pero cuya validez es difícil de justificar ante la comunidad científica.

6 Bibliografía y referencias

1. *Análisis y Gestión de Riesgos con MAGERIT en la Dirección Regional de Sistemas de Información y Telecomunicaciones. Contrato CARMMA 3142-UMU. CARM (Comunidad Autónoma de la Región de Murcia) y GIS (Grupo de Investigación de Ingeniería del Software, Universidad de Murcia)*. 1999.
2. *Manifesto for Agile Software Development*. 2001. Último acceso: Septiembre 2009. http://agilemanifesto.org.
3. *Technological transfer of a patient management system for the intensive care unit, contrato entre la Universidad de Murcia y la Fundación para la Investigación Biomédica en el Hospital de Getafe (Madrid).* 2006-2007.
4. *Software Technology Roadmap. Software Engineering Institute. Carnegie Mellon University*. 2008. Último acceso: Abril 2009. http://www.sei.cmu.edu/str/str.pdf.
5. *ARIS Suite, IDS Scheer*. 2009. Último acceso: Septiembre 2009. http://www.ids-scheer.com/international/en.
6. *Caliber-RM, Borland*. 2009. Último acceso: Septiembre 2009. http://www.borland.com/caliber/index.html.
7. *Caliber DefineIT, Borland*. 2009. Último acceso: Septiembre 2009. http://www.borland.com/us/products/caliber/index.html.
8. *Corporate Modeler Suite, Casewise*. 2009. Último acceso: Septiembre 2009. http://www.casewise.com.
9. *DOORS, Telelogic Rational (IBM Company)*. 2009. Último acceso: Septiembre 2009. http://www.telelogic.com/doors.
10. *Enterprise Architect, Sparx Systems Pty Ltd*. 2009. Último acceso: Septiembre 2009. http://www.sparxsystems.com.
11. *GEARS, BigLever Software*. 2009. Último acceso: Septiembre 2009. www.biglever.com/overview.html.
12. *i* Wiki*. 2009. Último acceso: Septiembre 2009. http://istar.rwth-aachen.de/tiki-view_articles.php.
13. *MetaCase - Domain-Specific Modeling with MetaEdit+*. 2009. Último acceso: Septiembre 2009. http://www.metacase.com/.
14. *Modeler, Holocentric*. 2009. Último acceso: Septiembre 2009. http://www.holocentric.com.
15. *ProS Labs: Moskitt Feature Modeler*. 2009. Último acceso: Septiembre 2009. http://www.pros.upv.es/mfm.
16. *ProVision, Metastorm*. 2009. Último acceso: Septiembre 2009. http://www.metastorm.com/products/provision_ea.asp.
17. *Requirements Management Tools, Ludwig Consulting Services*. 2009. Último acceso: Septiembre 2009. http://www.jiludwig.com/Requirements_Management_Tools.html.
18. *Requisite Web, IBM Rational Software.* 2009. Último acceso: Septiembre 2009. http://requisite.dif.um.es/reqweb.
19. *RequisitePro, IBM Rational Software.* 2009. Último acceso: Septiembre 2009. http://www-306.ibm.com/software/rational.
20. *The Reuse Company*. 2009. Último acceso: Septiembre 2009. http://www.reusecompany.com.
21. *Volere. Electronic Resources. The Atlantic Systems Guild*. 2009. Último acceso: Septiembre 2009. http://www.volere.co.uk/.
22. Alexander, I. *Requirements Engineering Tool Vendors and Freeware Suppliers*. 2009. Último acceso: Septiembre 2009. http://easyweb.easynet.co.uk/~iany/other/vendors.htm.
23. Alexander, I. y Kiedaisch, F. *Towards Recyclable System Requirements. 9th Annual IEEE Intl. Conf. and Workshop on the Engineering of Computer-Based Systems (ECBS'02).* 2002. Lund, Sweden.

24. Alrajeh, D., Russo, A., y Uchitel, S., *Inferring Operational Requirements from Scenarios and Goal Models Using Inductive Learning*. *Intl. Workshop on Scenarios and State Machines: Models, Algorithms, and Tools*. 2006, ACM: Shanghai, China.

25. Álvarez, B., Sánchez, P., Pastor, J. A., y Ortiz, F., *An Architectural Framework for Modeling Teleoperated Service Robots.* ROBOTICA - Intl. Journal of Information, Education and Research in Robotics and Artificial Intelligence, 2006. 24(4): p. 411-418.

26. Antón, A. I. *Publications Guide*. 2009. Último acceso: Septiembre 2009. http://www4.ncsu.edu/~aianton/publications.html.

27. Antón, A. I. y Potts, C. *The Use of Goals to Surface Requirements for Evolving Systems*. *20th Intl. Conf. on Software Engineering (ICSE'98)*. 1998. Kyoto, Japan: IEEE Computer Society.

28. Aranda, G. N., Vizcaíno, A., Cechich, A., Piattini, M., y Soto, J. P. *Una Metodología ara la Elicitación de Requisitos en Proyectos GSD. Actas de las XII Jornadas de Ingeniería del Software y Bases de Datos (JISBD'2007)*. 2007. Zaragoza.

29. Arlow, J., Emmerich, W., y Quinn, J. *Literate Modelling–Capturing Business Knowledge with the UML*. *«UML»'98: Beyond the Notation, 1st Intl. Workshop, Springer LNCS 1618*. 1998. Mulhouse, France.

30. Arlow, J. y Neustadt, I., *Enterprise Patterns and MDA: Building Better Software with Archetype Patterns and UML*. The Addison-Wesley Object Technology Series. 2004, Boston: Addison-Wesley.

31. ATL. *M2M/Atlas Transformation Language (ATL)*. 2009. Último acceso: Septiembre 2009. http://wiki.eclipse.org/ATL.

32. Barkerville, R., *Conducting Action Research: High Risk and High Reward in Theory and Practice*. En *Qualitative Research in Information Systems*, Trauth, E., Editor. 2001, Idea Group Publishing. p. 192-218.

33. Baskerville, R. L., *Information Systems Design Methods: Implications for Information Systems Development.* ACM Computing Surveys, 1993. 25(4): p. 375-414.

34. Baskerville, R. L., *Investigating Information Systems with Action Research.* Communications of the Association for Information Systems, 1999. 2(19): p. Article nº 4, 1-31.

35. Baskerville, R. L. y Wood-Harper, A. T., *A Critical Perspective on Action Research as a Method for Information Systems Research.* Journal of Information Technology, 1996. 11(3): p. 235-246.

36. Bass, L., Klein, M., y Bachmann, F., *Quality Attribute Design Primitives*. 2000, SEI tech. report, CMU/SEI-2000-TR-017, SEI (Software Engineering Inst.), Carnegie Mellon University: Pittsburgh, PA.

37. Bayer, J., Gerard, S., Haugen, O., Mansell, J., Moller-Pedersen, B., Oldevik, J., Tessier, P., Thibault, J.-P., y Widen, T., *Consolidated Product Line Variability Modeling*. En *Software Product Lines. Research Issues in Engineering and Management*, Käkölä, T. y Dueñas, J.C., Editores. 2006, Springer: Berlin Heidelberg. p. 195-241.

38. Berenbach, B. *The Automated Extraction of Requirements from UML Models*. *11th Intl. Conf. on Requirements Engineering (RE'03)*. 2003. Monterey, CA, USA: IEEE Computer Society.

39. Berenbach, B. *Comparison of UML and Text Based Requirements Engineering*. *Companion to the 19th Conf. on OO Programming, Systems, Languages, and Applications (OOPSLA'04)*. 2004. Vancouver, BC, Canada: ACM Press.

40. Berenbach, B. *Impact of Organizational Structure on Distributed Requirements Engineering Processes: Lessons Learned*. *1st Intl. Workshop on Global Software Development for the Practitioner (GSD 2006)*. 2006. Shanghai, China.

41. Bhat, J. M., Gupta, M., y Murthy, S. N., *Overcoming Requirements Engineering Challenges: Lessons from Offshore Outsourcing.* IEEE Software, 2006. 23(5): p. 38-44.

42. Biolchini, J., Gomes Mian, P., Cruz Natali, A. C., y Horta Travassos, G., *Systematic Review in Software Engineering*. 2005, Systems Engineering and Computer Science Department tech. report, TR-ES 679/05, COPPE/UFRJ: Rio de Janeiro, Brasil.

43. Blackburn, J. D., Scudder, G. D., y Van Wassenhove, L. N., *Improving Speed and Productivity of Software Development: A Global Survey of Software Developers.* IEEE Transactions on Software Engineering, 1996. 22(12): p. 875-885.

44. Booch, G., Rumbaugh, J., y Jacobson, I., *The Unified Modeling Language User Guide*. 2nd ed. Object Technology Series. 2005, Reading, MA: Addison-Wesley.

45. Breaux, T. D. y Antón, A. I., *Analyzing Regulatory Rules for Privacy and Security Requirements.* IEEE Transactions on Software Engineering, 2008. 34(1): p. 5-20.

46. Brooks, F. P., *No Silver Bullet - Essence and Accidents in Software Engineering.* Computer, 1987. 20(4): p. 10-19.

47. Bühne, S., Halmans, G., Lauenroth, K., y Pohl, K., *Scenario-Based Application Requirements Engineering*. En *Software Product Lines: Research Issues in Engineering and Management*, Käkölä, T. y Dueñas, J.C., Editores. 2006, Springer: Berlin Heidelberg. p. 161-194.

48. Bühne, S., Lauenroth, K., y Pohl, K. *Why is it not Sufficient to Model Requirements Variability with Feature Models? Automotive Requirements Engineering (AURE04), co-located at RE04*. 2004. Nanzan University, Nagoya, Japan.

49. Burke, D. y Johannisson, K. *Translating Formal Software Specifications to Natural Language: a Grammar-Based Approach*. *Logical Aspects of Computational Linguistics Conf.* 2005. Bordeaux, France.

50. Cabral, G. y Sampaio, A., *Formal Specification Generation from Requirement Documents.* Electronical Notes in Theorical Computer Science, 2008. 195: p. 171-188.

51. Calefato, F. y Lanubile, F. *Using The Econference Tool For Synchronous Distributed Requirements Workshops*. *Intl. Workshop on Distributed Software Development (DiSD 2005)* 2005. Paris, France.

52. Cantor, M. *Rational Unified Process for Systems Engineering. Part III: Requirements Analysis and Design*. 2003. Último acceso: Septiembre 2009. http://www-128.ibm.com/developerworks/rational/library/content/RationalEdge/oct03/m_rupse_mc.pdf.

53. Caro, G., *Herramienta CARE para el Modelado de Requisitos de Líneas de Producto*. 2009, Proyecto informático. Director: J. Nicolás. Facultad de Informática. Universidad de Murcia.

54. Castano, S., Fugini, M. G., Martella, G., y Samarati, P., *Database Security*. 1994, Wokingham: Addison-Wesley.

55. Cerdá, L., *Herramienta CARE para la Integración de Requisitos Textuales y Modelos en Ambientes de Líneas de Productos*. 2009, Proyecto informático. Director: J. Nicolás. Facultad de Informática. Universidad de Murcia.

56. Cerón, R., Dueñas, J. C., Serrano, E., y Capilla, R. *A Meta-Model for Requirements Engineering in System Family Context for Software Process Improvement Using CMMI*. *Intl. Conf. on Product Focused Software Process Improvement (PROFES 2005)*. 2005. Oulu, Finland.

57. Clements, P. y Northrop, L., *Software Product Lines. Practices and Patterns*. SEI Series in Software Engineering. 2002, Boston: Addison-Wesley.

58. CMMI, *Capability Maturity Model Integration, v1.1*. 2002, SEI tech. report, CMU/SEI-2002-TR-028, SEI (Software Engineering Institute), Carnegie Mellon University: Pittsburgh, PA.

59. CMMI. *CMMI, Capability Maturity Model Integration, v.1.2*. 2006. Último acceso: Septiembre 2009. http://www.sei.cmu.edu/cmmi/index.cfm.

60. Cockburn, A., *Structuring Use Cases with Goals.* Journal of Object-Oriented Programming (JOOP), 1997. 10(5/7).

61. Cox, K., Phalp, K. T., Bleistein, S. J., y Verner, J. M., *Deriving Requirements from Process Models Via the Problem Frames Approach.* Information and Software Technology, 2005. 47(5): p. 319-337.

62. Crofts, M., Smith, R., y Fraunholz, B. *Global Software Development: The Next RE Frontier? 9th Australian Workshop on Requirements Engineering (AWRE'04)*. 2004. Adelaida, Australia.

63. Cybulsky, J. *Reusing Requirements Specifications: Review of Methods and Techniques*. *1st Australian Requirements Engineering Workshop (AWRE'96)*. 1996. Caulfield, Australia.

64. Cybulsky, J. *Reuse of Early Life-Cycle Artifacts: Reusing Requirements with a Word Processor? 8th Annual Workshop on Institutionalizing Software Reuse (WISR)*. 1997. Columbus, USA.

65. Cybulsky, J. *Patterns in Software Requirements Reuse*. *3rd Australian Conf. on Requirements Engineering* 1998. Geelong, Australia.

66. Cybulsky, J., Neal, R., Kram, A., y Allen, J., *Reuse of Early Life-Cycle Artifacts: Workproducts, Methods and Tools.* Annals of Software Engineering, 1998. 5(1): p. 227-251.

67. Cybulsky, J. y Reed, K. *Requirements Classification and Reuse: Crossing Domains Boundaries*. *6th Intl. Conf. on Software Reuse (ICSR'2000)*. 2000. Vienna, Austria: Springer.

68. Chastek, G., Donohoe, P., Kang, K., y Thiel, S., *Product Line Analysis: A Practical Introduction*. 2001, SEI tech. report, CMU/SEI-2001-TR-001, SEI (Software Engineering Inst.), Carnegie Mellon University: Pittsburgh, PA.

69. Chen, P., *The Entity-Relationship Model - Toward a Unified View of Data*. ACM Transactions on Database Systems, 1976. 1(1): p. 9–36.

70. Cheng, B. H. C. y Atlee, J. M. *Research Directions in Requirements Engineering. Future of Software Engineering (FOSE'07)*. 2007. Minneapolis, USA.

71. Chung, L. *Dealing with Security Requirements during the Development of Information Systems. 5th Intl. Conf. on Advanced Information Systems Engineering (CAISE'93)*. 1993. Berlin, Germany: Spring Verlag.

72. Chung, L., Nixon, B. A., Yu, E., y Mylopoulos , J., *Non-Functional Requirements in Software Engineering*. The Kluwer International Series in Software Engineering. 2000, Boston: Kluwer Academic Publishers.

73. Damian, D. *An Empirical Study of Requirements Engineering in Distributed Software Projects: is Distance Negotiation More Effective?" 8th Asia-Pacific Software Engineering Conference (APSEC 2001)*. 2001. Macao, China.

74. Damian, D. *The Study of Requirements Engineering in Global Software Development: as Challenging as Important*. *Intl. Conf. on Software Engineering (ICSE'02)*. 2002. Orlando, USA.

75. Damian, D., *The Use of a Multimedia Group Support System for Distributed Software Requirements Meetings.* 35th Hawaii Intl. Conf. on System Sciences (HICSS'02), 2002. Hawaii, USA.

76. Damian, D. *A Research Methodology in the Study of Requirements Negotiations in Geographical Distributed Software Teams*. *11th IEEE Intl. Requirements Engineering Conference (RE'03)*. 2003. Monterey, CA, USA.

77. Damian, D., *Stakeholders in Global Requirements Engineering: Lessons Learned from Practice.* IEEE Software, 2007. 24(2): p. 21-27.

78. Damian, D., Chisan, J., Allen, P., y Corrie, B. *Awareness Meets Requirements Management: Awareness Needs in Global Software Development. Intl. Workshop on Global Software Development (GSD'03)*. 2003. Portland, USA.

79. Damian, D., Lanubile, F., y Mallardo, T., *On the Need for Mixed Media in Distributed Requirements Negotiations.* IEEE Transactions on Software Engineering, 2008. 34(1): p. 116-132.

80. Damian, D. y Zowghi, D. *The Impact of Stakeholders' Geographical Distribution on Managing Requirements in a Multi-Site Organization*. *Intl. Requirements Engineering Conf. (RE'02)*. 2002. Essen, Germany.

81. Daniels, J., Botta, R., y Bahill, T. *A Hybrid Requirements Capture Process. INCOSE 15th Annual Intl. Symposium on Systems Engineering*. 2005. Rochester, NY.

82. Davis, A. M., *Just Enough Requirements Management: Where Software Development Meets Marketing*. 2005, New York, NY: Dorset House.

83. De Landtsheer, R., Letier, E., y van Lamsweede, A., *Deriving Tabular Event-based Specifications from Goal-Oriented Requirements Models.* Requirements Engineering Journal, 2004. 9(2): p. 104-120.

84. Decker, B., Ras, E., Rech, J., Jaubert, P., y Rieth, M., *Wiki-Based Stakeholder Participation in Requirements Engineering.* IEEE Software, 2007. 24(2): p. 28-35.

85. Djebbi, O. y Salinesi, C. *RED-PL, a Method for Deriving Product Requirements from a Product Line Requirements Model*. *19th Intl. Conf. on Advanced Information Systems Engineering (CAiSE'07)*. 2007. Trondheim, Norway.

86. Djebbi, O., Salinesi, C., y Diaz, D. *Deriving Product Line Requirements: the RED-PL Guidance Approach*. *Asia-Pacific Software Engineering Conf. (APSEC 2007)*. 2007. Nagoya, Japan.

87. Djebbi, O., Salinesi, C., y Fanmuy, G. *Industry Survey of Product Lines Management Tools: Requirements, Qualities and Open Issues*. *15th IEEE Intl. Requirements Engineering Conf. (RE '07)*. 2007. Delhi, India.

88. Duque, A., Lasheras, J., y Toval, A. *ECAPRIS: Metodología Ágil de Medición de Calidad y Productividad en PyMEs*. *XIV Jornadas de Ingeniería del Software y Bases de Datos (JISBD'09)*. 2009. San Sebastián.

89. Durán, A. *Herramienta REM (REquirements Management)*. 2004. Último acceso: Septiembre 2009. http://www.lsi.us.es/descargas/descarga_programas.php?id=3.

90. Durán, A., Bernárdez, B., Ruiz, A., y Toro, M. *A Requirements Elicitation Approach Based in Templates and Patterns*. *Workshop on Requirements Engineering (WER'99)*. 1999. Buenos Aires, Argentina.

91. Durán, A., Ruiz, A., Corchuelo, R., y Toro, M. *Applying XML technology in Requirements Verification*. *5º Workshop Iberoamericano de Ingeniería de Requisitos y Ambientes Software (IDEAS'2002)*. 2002. La Habana, Cuba.

92. Ecklund, E., Delcambre, L., y Freiling, M., *Change Cases: Use Cases that Identify Future Requirements.* ACM SIGPLAN Notices, 1996. 31(10): p. 342 - 358.

93. EFTCoR. *Environmentally Friendly and Cost-Effective Technology for Coating Removal. Fifth Framework Programme, European Community, Subprogram Growth ref. GRD2-2001-50004*. 2005. Último acceso: Septiembre 2009. www.dsie.upct.es.

94. EMF. *Eclipse Modeling Framework*. 2008. Último acceso: Septiembre 2009. http://www.eclipse.org/modeling/emf/.

95. Eriksson, H., *The Semantic-Document Approach to Combining Documents and Ontologies.* Intl. Journal of Human-Computer Studies, 2007. 65(7): p. 624-639.

96. Eriksson, M., Börstler, J., y Borg, K. *Marrying Features and Use Cases for Product Line Requirements Modeling of Embeded Systems*. *4th Conf. on Software Engineering Research and Practice in Sweden (SERPS'04)*. 2004.

97. Estay, C. y Pastor, J. *Improving Action Research in Information Systems with Project Management*. *2000 Americas Conf. on Information Systems*. 2000. Long Beach, CA.

98. Faulk, S. *Product-Line Requirements Specifications (PRS): an Approach and Case Study*. *Fith Intl. Symposium on Requirements Engineering (RE'01)*. 2001. Toronto, Canada.

99. Favaro, J., *Managing Requirements for Business Value.* IEEE Software, 2002. 19(2): p. 15-17.

100. Fenton, N. y Neil, M., *A Strategy for Improving Safety Related Software Engineering Standards.* IEEE Transactions on Software Engineering, 1998. 24(11): p. 1002-1014.

101. Fernández, C., Iborra, A., Álvarez, B., Pastor, J. A., Sánchez, P., Fernández-Meroño, J. M., y Ortega, N., *Co-operative Robots in the Ship Repair Industry.* IEEE Robotics and Automation Magazine, 2005. 12(3): p. 65-77.

102. Finkelstein, A., *The Viewpoints FAQ.* Software Engineering Journal, 1996. 11: p. 2-4.

103. Finkelstein, A. y Emmerich, W. *The Future of Requirements Management Tools*. *Information Systems in Public Administration and Law*. 2000.

104. Firesmith, D., *Modern Requirements Specifications.* Journal of Object Technology, 2003. 2(1): p. 53-64.

105. Firesmith, D., *Generating Complete, Unambigous, and Verifiable Requirements from Stories, Scenarios, and Use Cases.* Journal of Object Technology, 2004. 3(10): p. 27-39.

106. Firesmith, D., *Specifying Reusable Security Requirements.* Journal of Object Technology, 2004. 3(1): p. 61-75.

107. FOSD. *Feature-Oriented Software Development Research*. 2009. Último acceso: Septiembre 2009. http://fosd.de/.

108. Fowler, M., *Analysis Patterns. Reusable Object Models*. 1997, Reading, MA: Addison Wesley.

109. French, W. L. y Bell, C. H., *Organizational Development: Behavioral Science Interventions for Organization Improvement*. 1996, London: Prentice Hall.

110. Gabb, A. *The Requirements Spectrum*. *First Regional Symposium of the Systems Engineering Society of Australia (SE'98)*. 1998. Canberra, Australia.

111. Gamma, E., Helm, R., Johnson, R., y Vlissides, J. M., *Design Patterns: Elements of Reusable Object-Oriented Software*. 1994, Boston: Addison-Wesley.

112. García-Molina, J., Ortín, M. J., Moros, B., y Nicolás, J., *Transforming the OOram Three Model Architecture into a UML-Based Process.* Journal of Object Technology, 2002. 1: p. 119-136.

113. García-Molina, J., Ortín, M. J., Moros, B., Nicolás, J., y Toval, A. *Towards Use Case and Conceptual Models through Business Modeling*. *19th Int. Conf. on Conceptual Modeling, ER'2000, LNCS 1920*. 2000. Salt Lake City, USA: Springer.

114. García, M., *Una Propuesta para la Actividad de Negociación de Requisitos en el Desarrollo Global de Software*. 2009, Proyecto informático. Director: J. Nicolás. Facultad de Informática. Universidad de Murcia.

115. Giese, H. y Wagner, R., *From Model Transformation to Incremental Bidirectional Model Synchronization*. Software and Systems Modeling, 2008. 8(1): p. 21-43.

116. Glass, R. L., *Software Engineering: Facts and Fallacies*. 2002, Boston: Addison-Wesley.

117. Glass, R. L., Vessey, I., y Ramesh, V., *Research in Software Engineering: an Analysis of the Literature.* Information and Software Technology, 2002. 44(8): p. 491-506.

118. GMF. *The Eclipse Graphical Modeling Framework*. 2008. Último acceso. http://www.eclipse.org/modeling/gmf.

119. Goguen, J. A. *The Dry and the Wet*. *IFIP TC8/WG8.1 Working Conf. on Information System Concepts: Improving the Understanding*. 1992.

120. Goguen, J. A. y Winkler, T., *Introducing OBJ3*. 1988, SRI tech. report, SRI-CSL-88-9: Menlo Park, CA.

121. Goldsmith, R. F., *Discovering Real Business Requirements for Software Project Success*. 2004, Boston, London: Artech House Publishers.

122. Gomaa, H., *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*. 2005, Boston: Addison-Wesley.

123. Gomaa, H. y Shin, M. *Multiple-View Meta-Modeling of Software Product Lines*. *8th Intl. Conf. on Engineering of Complex Computer Systems (ICECCS 2002)*. 2002. Greenbelt, USA.

124. González-Baixauli, B., Laguna, M. A., y Crespo, Y. *Product Line Requirements based on Goals, Features and Use Cases*. *Intl. Workshop on Requirements Reuse in System Family Engineering (IWREQFAM)*. 2004. Madrid.

125. Greenfield, J. y Short, K., *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. 2004, Indianapolis: Wiley.

126. Griss, M., Favaro, J., y d'Alessandro, M. *Integrating Feature Modeling with the RSEB*. *5th Intl. Conf. on Software Reuse*. 1998. Vancouver, Canada.

127. Guizzardi, G., Gerd, W., Guarino, N., y van Sinderen, M. *An Ontologically Well-Founded Profile for UML Conceptual Models*. *16th Intl. Conf. on Advanced Information Systems Engineering (CAiSE), LNCS 3084*. 2004. Riga, Latvia.

128. Gumm, D. *A Model of Requirements Engineering at Organizational Interfaces: An Empirical Study on Distributed Requirements Engineering*. *1st Intl. Global Requirements Engineering Workshop (GREW'07)*. 2007. Munich, Germany.

129. Halmans, G. y Pohl, K., *Communicating the Variability of a Software-Product Family to Customers.* Software and Systems Modeling, 2003. 2: p. 15-36.

130. Hanisch, J. y Corbitt, B. *Requirements Engineering During Global Software Development: Some Impediments to the Requirements Engineering Process – A Case Study*. *European Conf. on Information Systems (ECIS 04)*. 2004. Turku, Finland.

131. Hein, A., MacGregor, J., y Schlick, M. *Requirements and Feature Management for Software Product Lines*. *Deutscher Software-Produktlinien Workshop (DSPL-1)*. 2000. Kaiserslautern, Germany.

132. Heindl, M. y Biffl, S. *Risk Management with Enhanced Tracing of Requirement Rationale in Highly Distributed Projects*. *1st Intl. Workshop on Global Software Development for the Practitioner (GSD 2006)*. 2006. Shanghai, China.

133. Heindl, M., Reinisch, F., y Biffl, S. *Requirements Management Infrastructures in Global Software Development. Towards Application Lifecycle Management with Role-Oriented In-Time Notification*. *Intl. Workshop on Tool Support and Requirements Management in Distributed Projects (REMIDI'07)*. 2007. Munich, Germany.

134. Helén, M., *Challenges in Multi-Site and Multi-Cultural Globally Distributed Software Development.* Information System Science Bachelor Thesis. University of Jyväskylä, 2004.

135. Hernández, J., *Generación y Sincronización de Requisitos Textuales desde Modelos de Características*. 2009, Proyecto informático. Directores: J. Nicolás, A. Toval. Facultad de Informática. Universidad de Murcia.

136. Iborra, A., Caceres, D. A., Ortiz, F. J., Franco, J. P., Palma, P. S., y Alvarez, B., *Design of Service Robots.* IEEE Robotics & Automation, 2009. 16(1): p. 24-33.

137. IEEE, *Std. 610-1990, Glossary of Software Engineering Terminology*. 1990, The Institute of Electrical and Electronics Engineers, Inc. IEEE Software Engineering Stds. Collection.

138. IEEE, *Std 830-1998, Guide to Software Requirements Specifications*. *Volume 4: Resource and Technique Standards*. 1999, The Institute of Electrical and Electronics Engineers, Inc. IEEE Software Engineering Stds. Collection.

139. IEEE, *Std 1233-1998 Guide for Developing System Requirements Specifications*. *Volume 1: Customer and Terminology Standards*. 1999, The Institute of Electrical and Electronics Engineers, Inc. IEEE Software Engineering Stds. Collection.

140. IEEE/EIA, *Std 12207.0-1996. Industry Implementation of International Standard ISO/IEC 12207: 1995. Standard for Information Technology-Software Life Cycle Processes*. *Volume 1: Customer and Terminology Standards*. 1999, The Institute of Electrical and Electronics Engineers, Inc. IEEE Software Engineering Stds. Collection.

141. IEEE/EIA, *Std 12207.1-1997. Guide for Information Technology - Software life cycle processes - Life cycle data*. *Volume 1: Customer and Terminology Standards*. 1999, The Institute of Electrical and Electronics Engineers, Inc. IEEE Software Engineering Stds. Collection.

142. IKV. *Medini QVT*. 2008. Último acceso: Septiembre 2009. http://projects.ikv.de/qvt.

143. Illes-Seifert, T., Hermann, A., Geisser, M., y Hildenbrand, T. *The Challenges of Distributed Software Engineering and REquirements Engineering: Results of an Online Survey*. *1st Intl. Global Requirements Engineering Workshop (GREW'07)*. 2007. Munich, Germany.

144. INCOSE. *International Council on Systems Engineering*. *Requirements Management Tools Survey*. 2009. Último acceso: Septiembre 2009. http://www.incose.org/ProductsPubs/products/toolsdatabase.aspx.

145. IRqA. *IRqA, Visure Solutions*. 2009. Último acceso: Septiembre 2009. http://www.visuresolutions.com/products/irqa/irqa.php.

146. ISO, *ISO/IEC 7498-2:1989 Information Processing Systems – Open Systems Interconnection – Basic Reference Model – Part 2: Security Architecture*. 1989.

147. ISO, *ISO/IEC 15504 - Information Technology - Process Assessment*. 2004.

148. ISO, *ISO/IEC 27000 Information Technology - Security Techniques - Information Security Management Systems - Requirements*. 2005.

149. ISO, *ISO 9001:2008. Quality Management Systems -- Requirements*. 2008.

150. ISO/IEC, *ISO/IEC 15408. Evaluation Criteria for Information Technology Security*. 1999.

151. ITEA-Office. *ITEA Technology Roadmap for Software Intensive Systems (2nd Edition)*. 2004. Último acceso: Septiembre 2009. http://www.itea-office.org.

152. Jackson, M., *The Meaning of Requirements.* Annals of Software Engineering, 1997. 3(1): p. 5-21.

153. Jacobson, I., Christerson, M., Jonsson, P., y Övergaard, G., *Object Oriented Software Engineering: A Use Case Driven Approach*. 1992, Workingham, England: ACM Press. Addison Wesley.

154. John, I., *Capturing Product Line Information from Legacy User Documentation*. En *Software Product Lines. Research Issues in Engineering and Management*. 2006, Springer-Verlag: Berlin Heidelberg. p. 127-160.

155. John, I. y Muthig, D. *Product Line Modeling with Generic Use Cases*. *Intl. Workshop on Requirements Engineering for Product Lines (REPL'02)*. 2002. Essen, Germany.

156. Johnson, J., *Chaos: Charting the Seas of Information Technology*. 1994, Published Reports. Standish Group.

157. Jungmayr, S. y Stumpe, J. *Another Motivation for Usage Models: Generation of User Documentation*. *CONQUEST'98*. 1998. Nüremberg, Germany.

158. Kaindl, H., Brinkkemper, S., Bubenko, J. A., Farbey, B., Greenspan, S. J., Heitmeyer, C. L., Sampaio do Prado Leite, J. C., Mead, N. R., Mylopoulos, J., y Siddiqi, J. I. A., *Requirements Engineering and Technology Transfer: Obstacles, Incentives and Improvement Agenda.* Requirements Engineering Journal, 2002. 7(3): p. 113-123.

159. Käkölä, T. y Dueñas, J. C., eds. *Software Product Lines. Research Issues in Engineering and Management*. 2006, Springer: Berlin Heidelberg.

160. Kang, K., Cohen, S., Hess, J., Novak, W., y Peterson, A., *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. 1990, SEI tech. report, CMU/SEI-90-TR-021. SEI (Software Engineering Inst.), Carnegie Mellon University: Pittsburgh, PA.

161. Kang, K. C., Kim, S., Lee, J., Kim, K., Kim, G. J., y Shin, E., *FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures.* Annals of Software Engineering, 1998. 5(5): p. 143-168.

162. Kang, K. C., Kim, S., Lee, J., y Lee, K., *Feature-Oriented Engineering of PBX Software for Adaptability and Reusability.* Software—Practice & Experience, 1999. 29(10): p. 875–896.

163. Kitchenham, B. A., *Guidelines for performing Systematic Literature Reviews in Software Engineering*. 2007, EBSE tech. report, EBSE-2007-01. Software Engineering Group, School of Computer Science and Math., Keele University (UK), and Department of Computer Science, University of Durham (UK).

164. Kitchenham, B. A., Brereton, O. P., Budgen, D., Turner, M., Bailey, J., y Linkman, S., *Systematic Literature Reviews in Software Engineering - A Systematic Literature Review.* Information and Software Technology, 2009. 51(1): p. 7-15.

165. Kotonya, G. y Sommerville, I., *Requirements Engineering. Processes and Techniques*. 1998, New York, NY: Wiley.

166. Kwan, I., Damian, D., y Marczak, S. *The Effects of Distance, Experience, and Communication Structure on Requirements Awareness in Two Distributed Industria Software Projects*. *1st Intl. Global Requirements Engineering Workshop (GREW'07)*. 2007. Munich, Germany.

167. Kwintessential. *Intercultural Business Communication*. 2008. Último acceso: Septiembre 2009. http://www.kwintessential.co.uk/intercultural-business-communication/tool.php.

168. Laguna, M. A., López, O., y Crespo, Y. *Reuse, Standarization, and Transformation of Requirements*. *8th Intl. Conf. on Software Reuse (ICSR 2004)*. 2004. Madrid.

169. Lam, W., *A Case-Study of Requirements Reuse Through Product Families.* Annals of Software Engineering, 1998. 5(1): p. 253-277.

170. Lam, W., Jones, S., y Britton, C. *Technology Transfer for Reuse: A Management Model and Process Improvement Framework*. *3rd Intl. Conf. on Requirements Engineering (ICRE'98)*. 1998. Colorado Springs, USA.

171. Lam, W., McDermid, J. A., y Vickers, A. J., *Ten Steps Towards Systematic Requirements Reuse.* Requirements Engineering Journal, 1997. 2(2): p. 102-113.

172. Lam, W., Whittle, D., McDermid, J., y Wilson, S. *An Integrated Approach to Domain Analysis and Reuse for Engineering Complex Systems*. *IEEE Symposium and Workshop on Engineering of Computer Based Systems (ECBS'96)* 1996. Friedrichshafen, Germany.

173. Laplante, P. A. y Neill, C. J., *"The Demise of the Waterfall Model Is Imminent" and Other Urban Myths.* ACM Queue 2004. 1(1).

174. Larman, C., *Applying UML and Patterns*. 3rd ed. 2005, Upper Saddle River, N.J.: Prentice Hall.

175. Lasheras, J., *Análisis y Prototipado de una herramienta CARE de Soporte al Método SIREN: SirenTool*. 2003, Proyecto informático. Directores: A. Toval, J. Nicolás, B. Moros. Facultad de Informática. Universidad de Murcia.

176. Lasheras, J., Nicolás, J., Toval, A., y Moros, B. *Hacia un Modelo del Dominio de los Sistemas Teleoperados a Través de una Extensión de SIREN*. *II Jornadas de Trabajo DYNAMICA*. 2004. Málaga.

177. Lasheras, J., Toval, A., Nicolás, J., y Moros, B. *Soporte Automatizado a la Reutilización de Requisitos*. *VIII Jornadas de Ing. del Software y Bases de Datos (JISBD'03)*. 2003. Alicante.

178. Lasheras, J., Toval, A., Nicolás, J., y Moros, B. *Definición de Requisitos de Seguridad con Fines de Reutilización*. *Primer Taller de Seguridad en Ingeniería del Software y Bases de Datos*. 2004. Málaga.

179. Lavoie, B., Rambow, O., y Reiter, E. *The ModelExplainer*. *8th Intl. Workshop on Natural Language Generation (INLG'96)*. 1996. Herstmonceux Castle, England.

180. Lee, J. y Kang, K. C. *Feature Binding Analysis for Product Line Component Development*. *5th Intl. Workshop on Software Product-Family Engineering (PFE 2003), LNCS 3014*. 2004: Springer-Verlag Berlin Heidelberg.

181. Lee, K., Kang, K. C., Chae, W., y Choi, B. W., *Feature-Based Approach to Object-Oriented Engineering of Applications for Reuse.* Software—Practice & Experience, 2000. 30(9): p. 1025-1046.

182. Lee, K., Kang, K. C., y Lee, J. *Concepts and Guidelines of Feature Modeling for Product Line Software Engineering*. *7th Intl. Conf. on Software Reuse. LNCS 2319*. 2002. Austin, Texas: Springer-Verlag Berlin Heidelberg.

183. Letier, E. y van Lamsweede, A. *Deriving Operational Software Specifications from System Goals*. *10th Symposium on Foundations of Software Engineering 2002 (FSE'02)*. 2002. Charleston, South Carolina, USA: ACM Press.

184. LOPD, *Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal en España.* BOE, 1999. 298.

185. López, A., *SIRENgsd: Una Propuesta para la Reutilización de Requisitos en el Desarrollo Global de Software*. 2008, Proyecto informático. Director: J. Nicolás. Facultad de Informática. Universidad de Murcia.

186. López, A., Nicolás, J., y Toval, A. *A Repository of Risks and Safeguards for the Requirements Engineering Process in Global Software Development.* 2008. Último acceso: Septiembre 2009. http://www.um.es/giisw/GSD/.

187. López, A., Nicolás, J., y Toval, A. *Risks and Safeguards for the Requirements Engineering Process in Global Software Development*. *KNOWledge engINeering in Global software development (KNOWING), co-located with ICGSE 2009 (4th IEEE Intl. Conf. on Global Software Engineering)*. 2009. Limerick, Ireland.

188. Lutz, R., *Extending the Product Familiy Approach to Support Safe Reuse.* The Journal of Systems and Software, 2000. 53: p. 207-217.

189. MacGregor, E., Hsieh, Y., y Kruchten, P., *Cultural Patterns in Software Process Mishaps: Incidents in Global Projects.* HSSE 2005, 2005.

190. Madrigal, M., *Herramienta CARE de Soporte a SIRENgsd*. 2009, Proyecto informático. Director: J. Nicolás. Facultad de Informática. Universidad de Murcia.

191. Maiden, N., Jones, S., Ncube, C., y Lockerbie, J., *Using i* in Requirements Projects: Some Experiences and Lessons*. En *Social Modeling for Requirements Engineering*, Yu, E., Editor. 2007, MIT Press.

192. Maiden, N., Minocha, S., Manning, K., y Ryan, M. *CREWS-SAVRE: Systematic Scenario Generation and Use*. *3rd Intl. Conf. on Requirements Engineering (ICRE'98)*. 1998. Colorado Springs, CO, USA: IEEE Computer Society.

193. Maiden, N., Ncube, C., Kamali, S., Seyff, N., y Grünbacher, P. *Exploring Scenario Forms and Ways of Use to Discover Requirements on Airports that Minimize Environmental Impact*. *15th Intl. Requirements Engineering Conf. (RE'07)*. 2007. New Delhi, India.

194. Maiden, N. y Robertson, S. *Developing Use Cases and Scenarios in the Requirements Process*. *27th Intl. Conf. on Software Engineering (ICSE '05)*. 2005. St. Louis, MO, USA: ACM Press.

195. Maiden, N. y Sutcliffe, A., *Analogical Retrieval in Reuse-Oriented Requirements Engineering*. Software Engineering Journal, 1996. 11(5): p. 174-196.

196. Maiden, N. A. M., Manning, S., Jones, S., y Greenwood, J., *Generating Requirements from Systems Models Using Patterns: A Case Study*. Requirements Engineering Journal, 2005. 10(4): p. 276-288.

197. Mannion, M. y Kaindl, H., *Using Parameters and Discriminants for Product Line Requirements*. Systems Engineering, 2008. 11(1): p. 61-80.

198. Mannion, M., Lewis, O., Kaindl, H., Montroni, G., y Wheadon, J. *Representing Requirements on Generic Software in an Application Family Model*. *Intl. Conf. on Software Reuse (ICSR6)*. 2000. Vienna, Austria.

199. Mannion, M., Road, C., Kaindl, H., Wheadon, J., y Keepence, B. *Reusing Single System Requirements from Application Family Requirements*. *21st Intl. Conf. on Software Engineering (ICSE'99)*. 1999. Los Angeles, USA.

200. MAP, *MAGERIT –versión 1.0, Metodología de Análisis y Gestión de Riesgos de los Sistemas de Información*. 1996.

201. MAP. *MÉTRICA – versión 3. Metodología de Planificación, Desarrollo y Mantenimiento de Sistemas de Información*. 2001. Último acceso: Septiembre 2009. http://www.csae.map.es/csi/metrica3/index.html.

202. MAP. *MAGERIT – versión 2. Metodología de Análisis y Gestión de Riesgos de los Sistemas de Información*. 2006. Último acceso: Septiembre 2009. http://www.csi.map.es/csi/pg5m20.htm.

203. Martin, A., *What Drives the Configuration of Information Technology Projects? Exploratory Research in 10 Organizations.* Journal of Information Technology, 2003. 18(1): p. 1-15.

204. Martin, J. J. y Odell, J., *Métodos Orientados a Objetos: Conceptos Fundamentales*. 2ª ed. 1997, México: Prentice Hall.

205. Martínez, M. A., *Revisión de los Perfiles SIREN de Protección de Datos Personales y de Seguridad y Aplicación a un Caso de Estudio Real*. 2005, Proyecto informático. Directores: A. Toval, J. Nicolás, B. Moros. Facultad de Informática. Universidad de Murcia.

206. Martínez, M. A., Lasheras, J., Nicolás, J., y Toval, A. *Aplicación de un Proceso de Auditoría de Datos Personales Basado en el Método SIREN*. *III Jornadas de Trabajo DYNAMICA*. 2005. Almagro (Ciudad Real).

207. Martínez, M. A., Lasheras, J., Nicolás, J., y Toval, A. *Un Proceso de Auditoría de Datos Personales Basado en Ingeniería de Requisitos*. *I Simposio sobre Seguridad Informática (incluido en CEDI 2005)*. 2005. Granada.

208. Martínez, M. A., Lasheras, J., Toval, A., Fernández-Medina, E., y Piattini, M., *Systematizing the Personal Data Audit through Requirements Engineering in Health Information Systems: A Practical Case (aceptado, pendiente revisiones).* European Journal of Information Systems, 2007.

209. Martínez, M. Á., Lasheras, J., Toval, A., y Piattini, M. *An Audit Method of Personal Data Based on Requirements Engineering*. *4th Intl. Workshop on Security in Information Systems (WOSIS 2006)*. 2006. Paphos, Cyprus: INSTICC Press.

210. Matinlassi, M. *Comparison of Software Product Line Architecture Design Methods: COPA, FAST, FORM, KobrA and QADA. 26th Intl. Conf. on Software Engineering (ICSE'04)*. 2004. Edinburgh, Scotland.

211. Matulevicius, R., *Process Support for Requirements Engineering. A Requirements Engineering Tool Evaluation Approach*. 2005, Department of Computer and Information Science. Faculty of Information Technology, Mathematics and Electrical Engineering. Norwegian University of Science and Technology: Trondheim, Noruega.

212. Mavin, A. y Maiden, N. *Determining Socio-Technical Systems Requirements: Experiences with Generating and Walking Through Scenarios. 11th Intl. Conf. on Requirements Engineering (RE'03)*. 2003. Monterey, CA, USA: IEEE Computer Society.

213. Mellado, D., Fernández-Medina, E., y Piattini, M. *Security Requirements Variability for Software Product Lines. 3rd Intl. Conf. on Availability, Reliability and Security (ARES 2008)*. 2008. Barcelona.

214. Meyer, B., *Construcción de Software Orientado a Objetos*. 1999, Madrid: Prentice Hall.

215. Meziane, F., Athanasakis, N., y Ananiadou, S., *Generating Natural Language Specifications from UML Class Diagrams*. Requirements Engineering Journal, 2008. 13(1): p. 1-18.

216. Mili, H., Mili, F., y Mili, A., *Reusing Software: Issues and Research Directions*. IEEE Transactions on Software Engineering, 1995. 21(6): p. 528-562.

217. Monzón, A. y Dueñas, J. C. *Experience-based Approach to Requirements Reuse in Product Families with DOORS. Intl. Workshop on Requirements Reuse In System Family Engineering*. 2004. Madrid.

218. Moros, B., Nicolás, J., García-Molina, J., y Toval, A., *Combining Formal Specifications with Design by Contract*. JOOP (Journal of Object-Oriented Programming), 2000. 12(9): p. 16-22.

219. Moros, B., Vicente-Chicote, C., y Toval, A. *REMM-Studio+: Modeling Variability to Enable Requirements Reuse. 27th Conf. on Conceptual Modeling (ER 2008). LNCS 5231*. 2008. Barcelona.

220. Neighbors, J., *The Draco Approach to Constructing Software from Reusable Components*. IEEE Transactions on Software Enginering, 1984. 10(5): p. 564-573.

221. Neill, C. J. y Laplante, P. A., *Requirements Engineering: The State of the Practice*. IEEE Software, 2003. 20(6): p. 40-45.

222. Nicolás, J., Lasheras, J., Toval, A., Moros, B., Sánchez, P., y Álvarez, B. *Ingeniería de Requisitos Basada en Reutilización: Una Propuesta de Aplicación a los Sistemas Teleoperados para Limpieza de Cascos de Buques. III Jornadas de Trabajo DYNAMICA*. 2005. Almagro (Ciudad Real).

223. Nicolás, J., Lasheras, J., Toval, A., Ortiz, F. J., y Álvarez, B. *Una Experiencia de Modelado de los Sistemas Teleoperados para Limpieza de Cascos de Buques Mediante Características y Casos de Uso Genéricos. IV Jornadas de Trabajo DYNAMICA*. 2005. Archena (Murcia).

224. Nicolás, J., Lasheras, J., Toval, A., Ortiz, F. J., y Álvarez, B. *A Collaborative Learning Experience in Modelling the Requirements of Teleoperated Systems for Ship Hull Maintenance. Learning Software Organizations + Requirements Engineering (LSO+RE 2006)*. 2006. Hannover, Germany.

225. Nicolás, J., Lasheras, J., Toval, A., Ortiz, F. J., y Álvarez, B., *An Integrated Domain Analysis Approach for Teleoperated Systems*. Requirements Engineering Journal, 2009. 14(1): p. 27-46.

226. Nicolás, J., Moros, B., Lasheras, J., y Toval, A., *SIREN: Un Método Práctico de Ingeniería de Requisitos Basado en Reutilización, Versión 2.0*. 2009, DIS tech. report, TR DIS 1-2009, Departamento de Informática y Sistemas, Universidad de Murcia.

227. Nicolás, J. y Toval, A., *Gestión de Requisitos*. En *Fábricas de Software: Experiencias, Tecnologías y Organización*, Piattini, M. y Garzás, J., Editores. 2007, Ra-Ma: Madrid. p. 143-176.

228. Nicolás, J. y Toval, A., *Gestión de Requisitos*. En *Fábricas de Software: Experiencias, Tecnologías y Organización (2ª edición) (Publicación estimada otoño 2009)*, Piattini, M. y Garzás, J., Editores. 2009, Ra-Ma: Madrid.

229. Nicolás, J. y Toval, A., *On the Generation of Requirements Specifications from Software Engineering Models: A Systematic Literature Review*. Information & Software Technology, 2009. 51(9): p. 1291-1307.

230. Nicolás, J., Toval, A., Arenas, A., y Alcalde, J., *Formal Validation and Verification of Atomic Resolution Microscope Control and Topography*. Cybernetics and Systems, 2001. 32(8): p. 851-870.

231. Nuseibeh, B. y Easterbrook, S. *Requirements Engineering: A Roadmap*. *22nd Intl. Conf. on Software Engineering (ICSE'00)*. 2000. Limerick, Ireland: IEEE Computer Society Press.

232. Olivé, A. *On the Role of Conceptual Schemas in Information Systems Development*. *Ada-Europe, LNCS 3063*. 2004. Palma de Mallorca, Spain: Springer-Verlag Heidelberg.

233. OMG. *Model Driven Architecture*. 2003. Último acceso: Septiembre 2009. http://www.omg.org/mda/.

234. OMG. *MOF Meta Object Facility Specification. Object Management Group*. 2006. Último acceso: Septiembre 2009. http://www.omg.org/docs/formal/06-01-01.pdf.

235. OMG. *MOF QVT Final Adopted Specification. Object Management Group*. 2007. Último acceso: Septiembre 2009. http://www.omg.org/docs/ptc/07-07-07.pdf.

236. OMG. *MOF/XMI Mapping 2.1.1*. 2007. Último acceso: Septiembre 2009. http://www.omg.org/docs/formal/07-12-01.pdf.

237. OMG. *Software & Systems Process Engineering Meta-Model Specification (SPEM) v2*. 2008. Último acceso: Septiembre 2009. http://www.omg.org/technology/documents/formal/spem.htm.

238. OMG. *OMG's MetaObject Facility (MOF), Object Management Group*. 2009. Último acceso: Septiembre 2009. http://www.omg.org/mof/.

239. Ortiz, F., Iborra, A., Marín, F., Álvarez, B., y Fernández-Meroño, J. M. *GOYA: A Teleoperated System for Blasting Applied to Ships Maintenance*. *3rd Intl. Conf. on Climbing and Walking Robots*. 2000. Madrid.

240. Ortiz, F. P., J.A.; Alvarez, B.; Iborra, A.; Ortega, N.; Rodriguez, D.; Conesa, C. *Robots for hull ship cleaning*. *IEEE Intl. Symposium on Industrial Electronics (ISIE 2007)*. 2007. Vigo.

241. Parnas, D., *On the Design and Development of Program Families*. IEEE Transactions on Software Engineering, 1976. 2(1): p. 1-9.

242. Perry, D. E., Sim, S. E., y Easterbrook, S. *Case Studies for Software Engineers*. *29th Annual IEEE/NASA Software Engineering Workshop - Tutorial Notes (SEW'05)*. 2005. Greenbelt, MA.

243. Perry, D. E., Sim, S. E., y Easterbrook, S. *Case Studies for Software Engineers*. *28th Intl. Conf. on Software Engineering (ICSE '06). Full Day Tutorial*. 2006. Shanghai, China: ACM Press.

244. Pfleeger, S. L., *Software Engineering: Theory and Practice*. 1998, Upper Saddle River, NJ: Prentice Hall.

245. Piattini, M., Calvo-Manzano, J. A., Cervera, J., y Fernández, L., *Análisis y Diseño de Aplicaciones Informáticas de Gestión. Una Perspectiva de Ingeniería del Software*. 2004, Madrid: Ra-Ma.

246. Piattini, M. y Garzás, J., *Fábricas de Software: Experiencias, Tecnología y Organización*. 2007, Madrid: Ra-Ma.

247. Pisan, Y. *Extending Requirements Specifications Using Analogy*. *23th Intl. Conf. on Software Engineering (ICSE '00)*. 2000. Limerick, Ireland: ACM Press.

248. Pla, A., *Generación de Requisitos Textuales Incluyendo Literarios desde Modelos de Características y de Casos de Uso Genéricos*. 2009, Proyecto informático. Director: J. Nicolás. Facultad de Informática. Universidad de Murcia.

249. Pohl, K., Böckle, G., y van der Linden, F., *Software Product Line Engineering. Foundations, Principles and Techniques*. 2005, Berlin Heidelberg: Springer.

250. Pressman, R. S., *Ingeniería del Software. Un enfoque práctico*. 6ª ed. 2005, México: McGraw-Hill Interamericana.

251. Prieto-Diaz, R. y Arango, G., eds. *Domain Analysis and Software Systems Modelling*. 1991, IEEE Computer Society Press: Los Alamitos, CA.

252. Prikladinicki, R., Evaristo, R., Gallagher, K., Lopes, L. T., y Audy, J. L. N., *The Role of Culture in Interpreting Qualitative Data: Methodological Issues in an Exploratory Study of Cross-Cultural Distributed Software Development.* 13th Annual Cross-Cultural Meeting in Information Systems at ICIS, 2007.

253. Probasco, L. y Leffingwell, D., *Combining Software Requirements Specifications with Use Case Modeling*. *Rational white paper*. 1999.

254. PuLSE. *PuLSE (Product Line Software Engineering)*. 2009. Último acceso: Septiembre 2009. http://www2.iese.fraunhofer.de/Products_Services/pulse/.

255. Rabiser, R., Dhungana, D., Grunbacher, P., Lehner, K., y Federspiel, C. *Involving Non-Technicians in Product Derivation and Requirements Engineering: A Tool Suite for Product Line Engineering*. *15th IEEE Intl. Requirements Engineering Conf. (RE '07)*. 2007. Delhi, India.

256. Ramesh, B. y Jarke, M., *Toward Reference Models for Requirements Traceability*. IEEE Transactions on Software Engineering, 2001. 27(1): p. 58 -93.

257. Rashid, A. *Aspect-Oriented Requirements Engineering: An Introduction*. *16th Intl. Requirements Engineering Conf.* 2008. Barcelona.

258. Regnell, B., Kamsties, E., y Gervasi, V. *Summary of the 10th Anniversary Workshop on Requirements Engineering: Foundation for Software Quality*. *Requirements Engineering: Foundation for Software Quality 2004 (REFSQ'04)*. 2004. Riga, Latvia.

259. Reifer, D. J., *Is the Software Engineering State of the Practice Getting Closer to the State of the Art?* IEEE Software, 2003. 20(6): p. 78-83.

260. RequisitePro. *RequisitePro, IBM Rational Software.* 2009. Último acceso: Septiembre 2009. http://www-306.ibm.com/software/rational.

261. Riebisch, M., Böllert, K., Streitferdt, D., y Philipow, I. *Extending Feature Diagrams with UML Multiplicities*. *6th Conf. on Integrated Design and Process Technology (IDPT'2002)*. 2002. Pasadena, USA.

262. Rine, D. C. y Nada, N., *An Empirical Study of a Software Reuse Reference Model.* Information and Software Technology, 2000. 42(1): p. 47-65.

263. RMS, *Real Decreto 994/1999, de 11 de junio, por el que se aprueba el Reglamento de medidas de seguridad de los ficheros automatizados que contengan datos de carácter personal.* BOE, 1999. 151: p. 24241.

264. Robertson, S., *Requirements Patterns Via Events/Use Cases*. *Atlantic Co. white paper*. 1996.

265. Robertson, S. y Robertson, J., *Mastering the Requirements Process*. 2nd. ed. 2006, New York, NY: Addison-Wesley.

266. Runeson, P. y Höst, M., *Guidelines for Conducting and Reporting Case Study Research in Software Engineering.* Empirical Software Engineering, 2009. 14(2): p. 131-164.

267. Sawyer, P., Sommerville, I., y Viller, S., *Requirements Process Improvement Through the Phased Introduction of Good Practice.* Software Process Improvement and Practice, 1997. 3(1): p. 19-34.

268. Schlick, M. y Hein, A. *Knowledge Engineering in Software Product Lines*. *European Conf. on Artificial Intelligence (ECAI 2000), Workshop on Knowledge-Based Systems for Model-Based Engineering*. 2000. Berlin, Germany.

269. Schmid, K. y John, I., *A Customizable Approach To Full-Life Cycle Variability Management.* Science of Computer Programming, Elsevier, 2004. 53(3): p. 259--284.

270. Seyff, N., Graf, F., Grünbacher, P., y Maiden, N. *The Mobile Scenario Presenter: A Tool for in situ Requirements Discovery with Scenarios*. *15th Intl. Requirements Engineering Conf. (RE'07).* 2007. New Delhi, India.

271. Sinnema, M., Deelstra, S., Nijhuis, J., y Bosch, J. *COVAMOF: A Framework for Modeling Variability in Software Product Families*. *3rd Software Product Line Conf. (SPLC 2004)*. 2004. Boston, MA, USA.

272. Smite, D., *Requirements Management in Distributed Projects.* Journal of Universal Knowledge Management, 2006.

273. Smith, S. L. *Patterned Prose for Automatic Specification Generation*. *Conf. on Human Factors in Computing Systems*. 1982. Gaithersburg, Maryland, USA: ACM Press.

274. Snook, C., Poppleton, M., y Johnson, I., *Rigorous Engineering of Product-Line Requirements: A Case Study in Failure Management.* Information and Software Technology, 2008. 50(1-2): p. 112-129.

275. Sommerville, I., *Ingeniería del Software*. 7ª ed. 2005, Madrid: Addison-Wesley.

276. Sommerville, I., *Software Engineering*. 8th ed. 2007, Boston: Pearson Education Limited.

277. Stevens, P. *Bidirectional Model Transformation in QVT: Semantic Issues and Open Questions*. *MoDELS. 10th Intl. Conf. on Model Driven Engineering Languages and Systems*. 2007. Nashville, USA.

278. Svahnberg, M., van Gurp, J., y Bosch, J., *A Taxonomy of Variability Realization Techniques.* Software—Practice & Experience, 2005. 35(8): p. 705-754.

279. Tavakoli, R. y Reiser, M. O. *Reusing Requirements: The Need for Extended Variability Models*. *Intl. Symposium on Fundamentals of Software Engineering (FSEN 2007)*. 2007. Tehran, Iran.

280. Toval, A., Moros, B., Nicolás, J., y Lasheras, J., *Eight Key Issues for an Effective Reuse-Based Requirements Process.* Intl. Journal of Computer Science and Software Engineering, 2008. 23(6): p. 373-386.

281. Toval, A., Nicolás, J., y Moros, B., *Requisitos Reutilizables de Seguridad en Sistemas de Información y Bases de Datos*. En *Seguridad en Bases de Datos*, Fernández, E., Piattini, M., y Serrano, M., Editores. 2001, Dintel: Madrid. p. 269-300.

282. Toval, A., Nicolás, J., y Moros, B. *Un Proceso de Ingeniería de Requisitos Basado en Reutilización*. *Jornadas de Ingeniería de Requisitos Aplicada (JIRA 2001)*. 2001. Sevilla.

283. Toval, A., Nicolás, J., y Moros, B., *SIREN: Un Proceso de Ingeniería de Requisitos Basado en Reutilización*. En *Applying Requirements Engineering*, Durán, A. y Toro, M., Editores. 2002, Catedral: Sevilla. p. 57-72.

284. Toval, A., Nicolás, J., Moros, B., y García, F., *Requirements Reuse for Improving Information Systems Security: A Practicioner's Approach.* Requirements Engineering Journal, 2002. 6(4): p. 205-219.

285. Toval, A., Olmos, A., y Piattini, M. *Legal Requirements Reuse: A Critical Success Factor for Requirements Quality and Personal Data Protection*. *IEEE Intl. Joint Conf. on Requirements Engineering (ICRE'02 and RE'02)*. 2002. Essen, Germany: IEEE Computer Press.

286. Trigaux, J.-C. y Heymans, P., *Modelling Variability Requirements in Software Product Lines: a Comparative Survey*. 2003, Computer Science Inst., University of Namur.

287. Türetken, O., Su, O., y Demirörs, O. *Automating Software Requirements Generation from Business Process Models*. *1st Conf. on the Principles of Software Engineering (PRISE'04)*. 2004. Buenos Aires, Argentina.

288. van der Linden, F., Schmid, K., y Rommes, E., *Software Product Lines in Action. The Best Industrial Practice in Product Line Engineering*. 2007, Berlin Heidelberg: Springer.

289. van der Massen, T. y Lichter, H. *Modeling Variability by UML Use Case Diagrams*. *Intl. Workshop on Requirements Engineering for Product Lines (REPL'02)*. 2002. Essen, Germany.

290. van Lamsweede, A. *Goal-Oriented Requirements Engineering: A Roundtrip from Research to Practice*. *12th Requirements Engineering Conf. 2004 (RE'04)*. 2004. Kyoto, Japan: IEEE Publishers.

291. van Lamsweede, A. y Willemet, L., *Inferring Declarative Requirements Specifications from Operational Scenarios.* IEEE Transactions on Software Engineering, 1998. 24(12): p. 1089-1114.

292. Volere. *Requirements Tools*. 2009. Último acceso: Septiembre 2009. http://www.volere.co.uk/tools.htm.

293. von Knethen, A., Paech, B., Kiedaisch, F., y Houdek, F. *Systematic Requirements Recycling Through Abstraction and Traceability*. *10th Intl. Requirements Engineering Conf. (RE'02)*. 2002. Essen, Germany.

294. Wadsworth, Y. *What is Participatory Action Research?* 1998. Último acceso: Septiembre 2009. http://www.scu.edu.au/schools/gcm/ar/ari/p-ywadsworth98.html.

295. Wiegers, K. E., *Read my lips: no new models!* IEEE Software, 1998. 15(5): p. 10-13.

296. Wiegers, K. E., *Software Requirements*. 2nd ed. 2003, Redmond, WA: Microsoft Press.

297. Yingfei, X., Dongxi, L., Zhenjiang, H., Takeichi, M., y Mei, H. *Towards Automatic Model Synchronization from Model Transformations. 22th IEEE/ACM Intl. Conf. on Automated Software Engineering*. 2007. Atlanta, USA.

298. Yourdon, E., *Análisis estructurado moderno*. 1993, México: Prentice-Hall.

299. Yu, E. *Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering*. *3rd IEEE Intl. Symposium on Requirements Engineering (RE'97)*. 1997. Washington D.C., USA.

300. Yu, E., Du Bois, P., Dubois, E., y Mylopoulos, J. *From Organization Models to System Requirements. A Cooperating Agents Approach*. *3rd Intl. Conf. on Cooperative Information Systems (CoopIS-95)*. 1995. Vienna, Austria.

301. Zachos, K., Maiden, N., y Tosar, A., *Rich Media Scenarios for Discovering Requirements*. IEEE Software, 2005. 22(5): p. 89-97.

302. Zave, P., *Classification of Research Efforts in Requirements Engineering*. ACM Computing Surveys, 1997. 29(4): p. 315-321.

303. Zowghi, D. y Damian, D. *An Insight into the Interplay Between Culture, Conflict and Distance in Globally Distributed Requirements Negotiations*. *36th Hawaii Intl. Conf. on System Sciences (HICSS'03)*. 2003. Hawaii, USA.

304. Zowghi, D. y Damian, D., *RE Challenges in Multi-site Software Development Organisations*. Requirements Engineering Journal, 2003. 8(3): p. 149-160.