GENERACIÓN AUTOMÁTICA DE SERVICIOS WEB A PARTIR DE MODELOS CONCEPTUALES

Memoria para optar al grado de Doctor en Informática

Marta Ruiz Server

UNIVERSIDAD POLITECNICA DE VALENCIA

Director:

Dr. Vicente Pelechano Ferragud

Departamento de Sistemas Informáticos y Computación

Mayo 2010

Generación Automática de Servicios Web a partir de Modelos Conceptuales

Universidad Politécnica de Valencia
Departamento de Sistemas Informáticos y Computación

Memoria presentada por
Marta Ruiz Server
para optar al grado de Doctor en Informática
por la Universidad Politécnica de Valencia

Director de la Tesis:
Dr. Vicente Pelechano Ferragud

Valencia, 29 de abril de 2010

A mis padres, a quien les debo todo lo que soy.
A Ricardo y a Jana, mi inspiración.

Resumen

Los servicios Web facilitan el acceso a la funcionalidad de las aplicaciones a través de Internet, facilitando la interoperabilidad entre servicios y aplicaciones, y permitiendo integrar la funcionalidad de distintas aplicaciones empresariales. Además, proporcionan estándares y mecanismos para llevar a cabo el comercio electrónico, convirtiendo la Web en un marco ideal para el desarrollo de aplicaciones distribuidas en prácticamente todos los dominios de aplicación. Los servicios Web constituyen el principal mecanismo para implementar las Arquitecturas Orientadas a Servicios (SOA).

La evolución natural de los métodos de producción de software, y de OO-Method / OOWS en particular, plantean la necesidad de mejorar el proceso de producción de software. Para mejorarlo, se debe de dotar a los métodos del soporte necesario para el desarrollo de aplicaciones Web sobre SOA proporcionando una clara estrategia de generación automática de código. Desde el punto de vista de la Ingeniería Dirigida por Modelos, estas aplicaciones se deben de generar automáticamente a partir de modelos. La generación automática debe poder dar soporte, de forma transparente, a las diferentes tecnologías existentes en el ámbito de los servicios Web en la actualidad.

Esta tesis presenta un método, dentro del ámbito de la Ingeniería Web, que diseña e implementa de manera automática servicios Web a partir de modelos conceptuales (modelos que representan el sistema independientemente de los detalles tecnológicos). Para conseguir la independencia de tecnología, el método se basa en los principios del Desarrollo de Software Dirigido por Modelos (MDD). De esta forma, el método utiliza los modelos que proporciona la propuesta OO-Method / OOWS, y mediante la aplicación de transformaciones de *Modelo-A-Texto* se obtiene el diseño e implementación de los servicios Web que publican la funcionalidad del sistema modelado.

De entre los modelos utilizados, cabe destacar los modelos de la etapa de Especificación de Requisitos, donde están definidas las tareas que desea realizar el usuario. Las operaciones que se diseñan cubren los aspectos de funcionalidad esperada por el usuario, recuperación y tratamiento de datos, gestión de usuarios, soporte a la navegación de la aplicación y soporte a la presentación. El método propuesto en esta tesis está soportado por una herramienta llamada `DISWOOM`. Esta herramienta ha sido desarrollada en el entorno Eclipse

utilizando MOFScript como lenguaje para implementar las transformaciones de *Modelo-A-Texto*. DISWOOM cubre el método presentado tanto en la etapa de diseño como en la de generación de código de los servicios Web.

Resum

Els serveis Web faciliten l'accés a la funcionalitat de les aplicacions a través d'Internet, facilitant la interoperabilitat entre serveis i aplicacions, i permetent integrar la funcionalitat de diferents aplicacions empresarials. A més, proporcionen estàndards i mecanismes per a portar a terme el comerç electrònic, convertint la Web en un marc ideal per al desenvolupament d'aplicacions distribuïdes en pràcticament tots els dominis d'aplicació. Els serveis Web constitueixen el principal mecanisme per a implementar les Arquitectures Orientades a Serveis (SOA).

L'evolució natural dels mètodes de producció de programari, i de OO-Method / OOWS en particular, plantegen la necessitat de millorar el procés de producció de programari. Per a millorar-lo, es deu dotar als mètodes del suport necessari per al desenvolupament d'aplicacions Web sobre SOA proporcionant una clara estratègia de generació automàtica de codi. Des del punt de vista de l'Enginyeria Dirigida per Models, aquestes aplicacions es deuen generar automàticament a partir de models. La generació automàtica ha de poder donar suport, de forma transparent, a les diferents tecnologies existents en l'àmbit dels serveis Web en l'actualitat.

Aquesta tesi presenta un mètode, dins de l'àmbit de l'Enginyeria Web, que dissenya i implementa de manera automàtica serveis Web a partir de models conceptuals (models que representen el sistema independentment dels detalls tecnològics). Per ha aconseguir la independència de tecnologia, el mètode es basa en els principis del Desenvolupament de Programari Dirigit per Models (MDD). D'aquesta forma, el mètode utilitza els models que proporciona la proposta OO-Method / OOWS, i mitjançant l'aplicació de transformacions de *Model-A-Text* s'obté el disseny i implementació dels serveis Web que publiquen la funcionalitat del sistema modelat.

Entre els models utilitzats, cal destacar els models de l'etapa d'Especificació de Requisits, on estan definides les tasques que desitja realitzar l'usuari. Les operacions que es dissenyen cobreixen els aspectes de funcionalitat esperada per l'usuari, recuperació i tractament de dades, gestió d'usuaris, suport a la navegació de l'aplicació i suport a la presentació. El mètode proposat en aquesta tesi està suportat per una eina anomenada `DISWOOM`. Aquesta eina ha

estat desenvolupada en l'entorn Eclipse utilitzant MOFScript com llenguatge per a implementar les transformacions de *Model-A-Text*. `DISWOOM` cobreix el mètode presentat tant en l'etapa de disseny com en la de generació de codi dels serveis Web.

Abstract

Web services provide access to application functionality over the Internet, making easier interoperability between services and applications, allowing to integrate functionality in different business applications. They also provide standards and mechanisms for achieving electronic e-commerce, turning the Web into an ideal framework for developing distributed applications in multiple application domains. Web services are the main mechanism to implement Service Oriented Architecture (SOA).

The natural evolution of software production methods, and the OO-Method / OOWS in particular, raise the need to improve the software production process. In order to improve it, the methods should be provided with the necessary support for developing Web applications on SOA, providing a clear strategy of automatic code generation. From the Model Driven Engineering point of view, these applications should be automatically generated from models. Automatic generation allows dealing with the different technologies that today exists in the field of Web services in a transparent way.

Within the context of the Web Engineering field, this thesis presents a method to automatically perform the design and implementation of Web services from conceptual models (models that represent the system independently of technological details). To achieve this technology independence, the method is based on the principles Model Driven Development (MDD). Thus, the method uses the models provided by the OO-Method / OOWS proposal, and it applies *Model-to-Text* transformations to obtain the design and implementation of Web services that publish the modeled system functionality.

Among the used models, it is important to emphasize the role of the models of the Requirements Specification stage, where the tasks that the user wants to perform are defined. The designed operations cover the aspects of the functionality expected by the user, the data retrieval, the user management, and the navigation and the presentation support.The method proposed in this thesis is supported by a tool called `DISWOOM`. This tool has been developed in the Eclipse environment by using MOFScript to implement the *Model-to-Text* transformations. `DISWOOM` gives support to the method in the design and code generation steps for Web service development.

Agradecimientos

Dice un refrán que "*es de bien nacidos el ser agradecidos*", y por eso no puedo dejar de dar las gracias a todos aquellos que, de una u otra manera, han hecho posible la elaboración de esta tesis doctoral y el trabajo de investigación necesario para su logro.

A mi director de tesis el Dr. Vicente Pelechano, por haberme dado la oportunidad de conocer el mundo de la investigación. Por sus consejos, ánimos, tiempo y sobre todo por la confianza que ha depositado en mí desde el día en que nos conocimos.

Al Dr. Oscar Pastor, por guiarme en los comienzos y darme la oportunidad de entrar en el grupo.

A Pedro, Gonzalo (¡te echamos de menos!), Vicky y Javi, con los que he compartido laboratorio, amistad y tantos buenos momentos.

A Pedro (de nuevo) y a Carlos, sin sus ánimos y su apoyo esta tesis no habría llegado a su fin.

A Manoli y Joan, por su ayuda en mis inicios como investigadora.

A Ricardo Quintero y Paco, por su apoyo incondicional.

A Pau, por su ayuda en los pasos a seguir para poder leer esta tesis.

A Enric, Ausiàs y Eva, a los que seguí para adentrarme en el mundo de la empresa privada.

Al resto de miembros del grupo OO-Method (ahora ProS), gracias por vuestro compañerismo.

También quiero agradecer su apoyo a mis compañeros en Indra Software Labs. En especial a Miguel Ángel, Carmen, Oriol, Javi, Manolito y Raúl, sin los buenos ratos que hemos pasado y el trabajo conjunto que hemos realizado, esta tesis no habría sido igual. He aprendido mucho de vosotros, pero sobre todo me habéis enseñado el verdadero valor de trabajar en equipo.

Gracias a todo el equipo de GESCEN y de SIA, vuestro compañerismo me ha ayudado a superar las piedras del camino. Gracias por vuestro apoyo.

En especial quiero agradecerles a mis padres Vicente y Juana, la oportunidad y los ánimos que me han dado para estudiar una carrera y llegar hasta el punto de escribir esta tesis.

A mis sobrinos Paula, Miguel, Dani, Alex, Juanjo y Sofía, por el tiempo que les debo.

A mis hermanas, Eva y Silvia, a mis cuñados Dani, Miguel, Javi, Rosa y José Antonio y por supuesto a mis suegros Paco y Dora, por estar siempre ahí.

A Ricardo, por su amor, dedicación, paciencia, por cuidarme y animarme a terminar esta tesis, pero sobre todo, por haberme dado a Jana. Sin ti, nunca hubiera podido terminar este trabajo.

A Jana, porque aprendes cada día algo nuevo y no me lo quiero perder. A partir de ahora todo mi tiempo es tuyo.

A todos mi más sincera gratitud.

Índice general

Índice de figuras

Índice de algoritmos

Índice de tablas

Capítulo 1

Introducción

1.1 Introducción

El trabajo desarrollado en esta tesis se sitúa en el contexto del modelado conceptual y la generación automática de servicios Web en arquitecturas orientadas a servicios (SOA [1]). El software ha dejado de ser una aplicación aislada y se ha convertido en un servicio ofrecido a usuarios u otros elementos software. En este contexto, los servicios se definen como un conjunto de tecnologías [2, 3] con capacidad para interoperar en la Web y que intercambian datos entre sí, con el objetivo de ofrecer unos servicios. En el caso de los servicios Web [4], las principales tecnologías en las que se fundamentan son XML [5], SOAP [6], WSDL [7] y UDDI [8].

Actualmente, se pueden encontrar diversas aplicaciones Web comerciales que hacen uso de estas tecnologías para publicar su funcionalidad a través de servicios Web. Los ejemplos más populares, utilizando el protocolo SOAP (según el directorio de servicios Web `programmableweb` [1]), son: `Flickr` [2], el cual proporciona servicios para compartir fotos; `Amazon` [3] proporciona un conjunto de servicios Web que permiten acceder a su catalogo completo de pro-

[1] http://www.programmableweb.com/apis/directory
[2] http://code.flickr.com
[3] http://aws.amazon.com

ductos; `Ebay` [4] publicita servicios Web para la compra online en subastas; y `Google` [5], que proporciona, entre otros, los servicios Web de `google search` o `google adSense`.

De este modo, se puede ver cómo el uso de servicios Web para publicar la funcionalidad de las aplicaciones Web, es hoy en día una realidad. Sin embargo, dentro de la comunidad de Ingeniería Web [9], no existe un amplio soporte para el desarrollo metodológico de este tipo de software.

La principal contribución del trabajo de esta tesis consiste en una aproximación para el desarrollo de servicios Web, que aplica el enfoque del Desarrollo de Software Dirigido por Modelos (DSDM) [10]. En concreto, se presenta un método para llevar a cabo la identificación y categorización de los servicios Web en el ámbito del modelado conceptual (espacio del problema) y su generación automática sobre una arquitectura SOA (espacio de la solución). Dicho método se ha desarrollado en el ámbito del método de producción automática de software OO-Method [11], y su extensión para la Web OOWS [12, 13].

Así pues, el método propuesto permite la generación automática de servicios Web a partir de los modelos definidos en la diferentes fases de desarrollo del método OO-Method / OOWS. Para ello, se llevan a cabo dos grandes pasos:

1. *Diseño de servicios Web a partir de los modelos OO-Method / OOWS.* En este paso se analizan los diferentes aspectos capturados en los modelos OO-Method / OOWS (lógica de aplicación, navegación, usuarios, etc.), y se crea una categorización funcional a partir de ellos, que constituye la base para el diseño de servicios Web. Además, se identifican las operaciones que pueden ser definidas en cada una de estas categorías, junto a los parámetros y tipos correspondientes, a partir de las primitivas conceptuales de OO-Method / OOWS.

2. *Estrategia de Implementación.* En este paso se define un conjunto de transformaciones de *Modelo-A-Texto* que, teniendo en cuenta la categorización de operaciones realizada, generan (a) el código WSDL necesario para publicar los servicios Web y (b) el código Java que implementa

[4] http://developer.ebay.com/
[5] http://code.google.com

la funcionalidad proporcionada por los mismos. Dicho código Java hace uso del código generado por las herramientas que dan soporte al método OO-Method/OOWS.

Por último, se introduce una herramienta que da soporte al método. Dicha herramienta, aplica de forma automática las transformaciones de *Modelo-A-Texto* que generan los servicios Web que dan soporte a un modelo OO-Method / OOWS específico. Para desarrollar esta herramienta se ha hecho uso del lenguaje de definición de transformaciones de *modelo-a-texto* MOFScript [14, 15].

A continuación, se presenta la motivación que ha llevado a la realización de esta tesis. En el tercer apartado se comentan las contribuciones de la tesis. En el cuarto apartado se presenta el entorno de trabajo en el que se ha desarrollado la tesis y a continuación se comenta la metodología de trabajo utilizada. Por último, se introduce la estructura de la memoria, en la que se describe el contenido de cada uno de los capítulos desarrollados.

1.2 Motivación

El interés por los servicios Web ha ido aumentando de forma progresiva a lo largo de los años. Prueba de ello, es el creciente uso de esta nueva tecnología en el desarrollo de aplicaciones Web comerciales (como por ejemplo, las aplicaciones introducidas anteriormente `Flickr`, `Amazon`, `Ebay` y `Google`). Desde su aparición, la comunidad investigadora en el ámbito de los servicios Web se ha centrado, principalmente, en dar soporte al desarrollo de aplicaciones que integran servicios Web en sus procesos de negocio [16, 17, 18]. En este sentido, podemos encontrar diversas aproximaciones en la Ingeniería Web (WebML [19, 20], OOHDM [21], UWE [22, 23], OO-H [22]) que facilitan al desarrollador la integración en sus aplicaciones de funcionalidad publicada por terceras partes mediante servicios Web .

En estas soluciones, el desarrollador es considerado como un consumidor de servicios. Sin embargo, no se tienen en cuenta adecuadamente a aquellos desarrolladores encargados de diseñar e implementar servicios Web para publicar funcionalidad que pueda ser consumida por otros. Así pues, el diseño e

implementación de servicios Web se lleva a cabo, mayoritariamente, mediante soluciones ad-hoc [24, 25, 26, 27], sin un soporte ingenieril, dificultando de este modo su desarrollo y posterior mantenimiento y extensión.

Ante esta situación, han aparecido en los últimos años nuevos trabajos y extensiones de otros ya existentes dentro del marco de la Ingeniería Web (Hera [28], OO-H [22], OOHDM [29], WebML [30]). Estos trabajos abordan el problema de proporcionar soporte metodológico para el diseño e implementación de servicios Web. Sin embargo, tal y como se estudiará en el capítulo 3, aún existen aspectos del desarrollo de servicios Web que estas aproximaciones no cubren completamente. Por ejemplo, aunque muchas de ellas proporcionan mecanismos para definir a nivel conceptual servicios Web, ninguna afronta el problema de proporcionar a los desarrolladores una herramienta que les facilite el proceso de identificar y diseñar los servicios que mejor se adecuan a los requisitos de su aplicación. Una de las principales causas de este problema es el hecho de que el soporte proporcionado por estas aproximaciones se centra principalmente en etapas de diseño e implementación, proporcionando poco o ningún soporte para afrontar el diseño de servicios Web desde etapas tempranas del desarrollo de software .

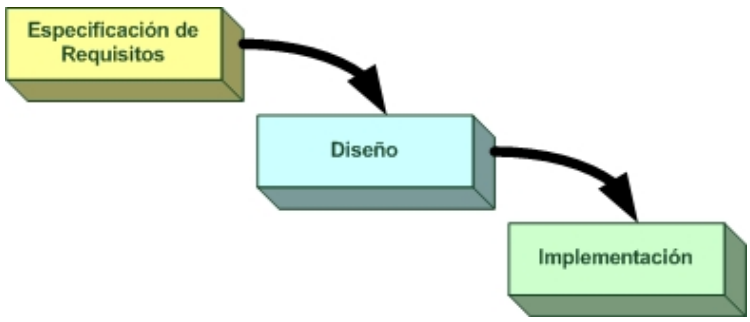

Figura 1.1: Principales etapas del ciclo de vida de los servicios Web

Así pues, si se considera que el desarrollo de servicios Web (desde cero) debe ser soportado, como cualquier otro tipo de software, en todas las fases del proceso de desarrollo [31] (*especificación de requisitos, diseño e implementación*, ver figura 1.1), es necesario introducir guías, herramientas y técnicas que permitan alcanzar dicho objetivo. Si se considera este objetivo desde la perspectiva de una de las aproximaciones de desarrollo de software más relevantes actualmente (Desarrollo de Software Dirigido por Modelos [10]), es necesario

introducir guías, herramientas y técnicas que nos permitan identificar y diseñar servicios Web, a partir de los modelos creados a lo largo del proceso de desarrollo (tanto aquellos centrados en la descripción de los requisitos de usuario, como aquellas enfocadas al diseño de software).

El desarrollo dirigido por modelos (Model Driven Development, MDD) [10, 32, 33] propone iniciar el proceso de desarrollo a partir de modelos de alto nivel de abstracción, transformando las abstracciones de estos modelos en primitivas de más bajo nivel, que en última instancia se podrán corresponder con líneas de código en un lenguaje de programación.

El trabajo de tesis se desarrolla en el contexto del método OO-Method, y su extensión para la Web, OOWS, que permiten la generación automática de código. Así pues, el trabajo proporciona soporte para el desarrollo de servicios Web mediante el uso de los modelos OO-Method / OOWS, definidos a lo largo de las diferentes etapas del proceso de desarrollo.

En la figura 1.2 se muestran los modelos de OO-Method / OOWS (*Especificación de Requisitos* y *Modelado Conceptual*) y el código que generan a partir de estos modelos (*Desarrollo de la solución*). En este trabajo de tesis se utilizan, de entre los modelos que ofrece la propuesta OO-Method / OOWS, los modelos de *Requisitos*, *Objetos*, *Usuarios*, *Navegacional* y *Presentación* marcados en dicha figura. Esto permite utilizar la etapa de *Especificación de Requisitos* como un modelo más a la hora de diseñar e implementar los servicios Web, con lo que se cubren las tres etapas principales, comentadas anteriormente, del ciclo de vida de los servicios Web. La propuesta convierte a los servicios Web en ciudadanos de primera clase para el método OO-Method/OOWS.

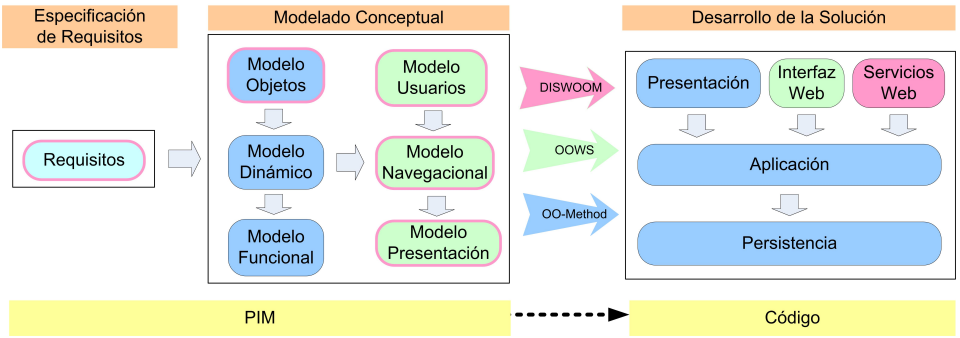

Figura 1.2: Modelos y espacio de la solución de la propuesta de tesis

Dado que el presente trabajo se desarrolla dentro de un ambiente de generación automática de código, se plantea el objetivo adicional de alcanzar el diseño e implementación de los servicios Web de manera automática a partir de estos modelos (la flecha nombrada como `DISWOOM` en la figura 1.2).

1.3 Contribuciones

La contribución principal de la tesis es "*definir un método que extienda el proceso de producción de software para incluir el diseño y la implementación de servicios Web*" (las contribuciones se puede ver con más detalle en los capítulos 4, 5, 6 y 7). El método permite: la identificación y especificación de servicios Web en las etapas de especificación de requisitos y modelado conceptual (Espacio del Problema) y su representación software (en el Espacio de la Solución) de forma automática. La principal contribución se lleva a cabo, en el contexto del método OO-Method / OOWS, con la consecución de las tres siguientes contribuciones específicas.

La primera contribución es "*la definición de un método para la identificación de las operaciones y servicios Web a partir de modelos navegacionales (también llamados diseños navegacionales)*". Esta contribución se lleva a cabo con la consecución de los siguientes objetivos:

- Estudiar los modelos proporcionados por OO-Method / OOWS con el fin de determinar cuáles ofrecen información relevante para la identificación de las operaciones de los servicios Web. Como novedad importante respecto a otros métodos, se utilizan los modelos de la etapa de *Especificación de Requisitos*.
- Identificar un conjunto de grupos funcionales a partir de las operaciones de los servicios Web detectadas. Este conjunto de grupos funcionales permiten agrupar operaciones de acuerdo con la funcionalidad que ofrecen.
- Definir de manera unívoca la relación existente entre los modelos OO-Method/ OOWS, los grupos funcionales, las operaciones y los servicios Web. Su objetivo es definir reglas que permitan realizar la identificación de operaciones de forma automática.

La segunda contribución es "*la definición de una estrategia de implementación de los servicios Web a partir de modelos navegacionales*". A continuación se detalla esta contribución:

- Estudiar los modelos proporcionados por OO-Method / OOWS con el fin de determinar qué primitivas, propiedades y relaciones conceptuales ofrecen información relevante para la implementación de las operaciones de los servicios Web.
- Definir una estrategia de implementación de los servicios Web detectados anteriormente a partir de la implementación generada por el método OO-Method / OOWS.

Estas dos contribuciones se materializan en "*la construcción de una herramienta que da soporte a las propuestas anteriores para automatizar el diseño e implementación de los servicios Web a partir de los modelos OO-Method / OOWS*". Se lleva a cabo con la consecución de los siguientes objetivos:

- Definir reglas de transformación *Modelo-A-Texto* para la generación automática de los servicios Web, definiendo un conjunto de correspondencias entre los modelos, la implementación generada por OO-Method / OOWS y la propuesta en este trabajo de tesis.
- Construir una herramienta CASE basada en el enfoque MDA que implemente las estrategias de diseño e implementación comentadas en este documento.

La última contribución es "*la definición del proceso de desarrollo OO-Method / OOWS, usando una técnica de descripción de procesos como SPEM v2.0 [34], y la integración del desarrollo de servicios Web en su proceso de desarrollo*". El proceso de desarrollo OO-Method / OOWS está formado por tres grandes etapas: *Especificación de Requisitos*, *Modelado Conceptual* y *Generación de Código*. A estas tres etapas se le añade la etapa de *Diseño e Implementación de servicios Web*. Esta etapa propone como actividades principales (1) la identificación de las operaciones que forman los servicios Web

a partir de la etapa de *Modelado Conceptual* y (2) la implementación de las mismas a partir del código generado en la etapa de *Generación de Código*.

La razón de elegir de OO-Method / OOWS en esta tesis, es el profundo conocimiento que el grupo de investigación, en el cual ha sido desarrollada la tesis, tiene de este método. Este aspecto ha facilitado un análisis exhaustivo de las primitivas de cada uno de los modelos y la reutilización del código generado por cada una de las herramientas que dan soporte a OO-Method / OOWS.

1.4 Entorno de la tesis

Este trabajo de tesis se ha desarrollado en el seno del grupo de investigación *OO-Method: Métodos de Producción del Software (Object-Oriented Methods for Software Development)*[6], perteneciente al Departamento de Sistemas Informáticos y Computación de la Universidad Politécnica de Valencia. El trabajo presentado en esta tesis forma parte de los trabajos realizados en el contexto de OO-Method / OOWS por un subgrupo de investigadores, en el cual la autora ha participado activamente.

La investigación que ha desembocado en el presente trabajo, ha sido realizada en su mayor parte con la ayuda de una beca de Formación de Personal Investigador (FPI), concedida por la Universidad Politécnica de Valencia. Conjuntamente, los trabajos que han posibilitado el desarrollo de esta tesis se enmarcan en los siguientes proyectos de investigación y desarrollo:

- "Desarrollo de E-Servicios para la nueva Sociedad Digital", proyecto DESTINO. Ministerio de Educación y Ciencia. D.G. de Investigación. Proyecto del Plan Nacional I+D+I 2004-2007, con ref. TIN2004-03534. Desde 2005 hasta 2007.

- "WEE-NET: Web Engineering Network of Excellence". European Comission (Alpha European Union Project). Desde 2004 hasta 2007.

[6] Actualmente el grupo de investigación se ha convertido en el Centro de Investigación en Métodos de Producción Software (ProS).

- "Ingeniería de Ambientes Web". Proyecto CICYT, con ref. TIC2001-3530-C02-01. Desde 2002 hasta 2004.
- "TAISSI: Tecnología Software Avanzada para la Ingeniería del Software de la Sociedad de la Información", Subproyecto "INGENIERÍA DE AMBIENTES WEB". Ministerio de Ciencia y Tecnología. D.G. De Investigación. Proyecto CICYT, con ref. TIC TIC2001-3530-C02-01. Desde 2001 hasta 2004.
- "WEST: Web-oriented Software Technology" Proyecto CYTED VII-18 (Programa Iberoamericano de Ciencia y Tecnología para el Desarrollo). Desde Agosto de 2000 hasta Agosto de 2003.

1.5 Metodología de Trabajo

La metodología seguida para llevar a cabo el plan de trabajo que ha permitido la consecución de los objetivos establecidos, se basa en la metodología de investigación descrita en [35] y en [36]. La investigación llevada a cabo en esta tesis está compuesta por cinco pasos: (1) identificación del problema, (2) propuesta, (3) desarrollo, (4) evaluación y (5) conclusión. En la figura 1.3 se muestran gráficamente estos pasos.

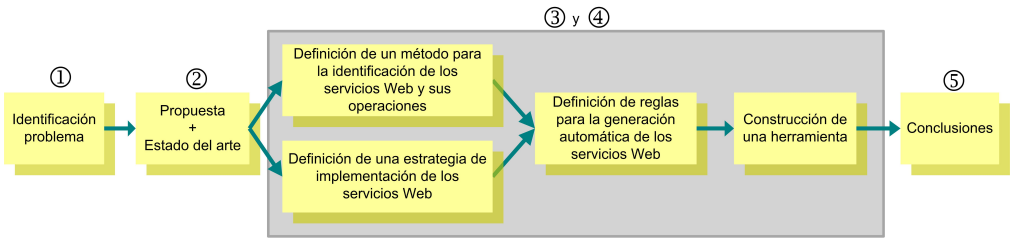

Figura 1.3: Etapas de la metodología de trabajo utilizada en la tesis

Se comienza, de acuerdo a la metodología, identificando el problema a resolver en la tesis. Como resultado, se identifican claramente los objetivos a cubrir para resolver este problema.

A continuación, se lleva a cabo el segundo paso que se corresponde con la propuesta para solucionar el problema y el estudio del estado del arte, donde

se estudian las propuestas más relevantes sobre diseño e implementación de servicios Web. Además, en esta tarea se estudiarán los conceptos, los fundamentos tecnológicos y los estándares (XML, SOAP, UDDI, WSDL) en los que se basan los servicios Web y las arquitecturas SOA.

Una vez descrita la solución propuesta, se llevan a cabo los pasos tres y cuatro:

- Primero, se define un método que permite identificar, de forma precisa, la funcionalidad a ofrecer por la aplicación mediante servicios Web a partir de los modelos OO-Method / OOWS.
- Paralelamente, se define una estrategia que permite identificar la correspondencia entre la funcionalidad que se genera automáticamente a partir de los modelos OO-Method / OOWS y aquella que forma parte de las operaciones ofrecidas en los servicios Web obtenidos en la fase anterior.
- En la siguiente fase, se define un proceso de transformación que establece las correspondencias entre los elementos de los modelos de OO-Method / OOWS, el código generado y los servicios Web identificados.
- Una vez las reglas han sido definidas, se implementan dichas reglas para crear una herramienta que permita obtener el código de los servicios Web.

Finalmente, en el paso cinco, se analizan los resultados del trabajo de investigación para obtener conclusiones y áreas de futuros trabajos.

1.6 Estructura de la Tesis

La tesis se estructura en los siguientes capítulos:

Capítulo 1. Introducción

El presente capítulo introduce en primer lugar el contexto en el que se desarrolla el tema de la tesis, seguidamente se presenta la motivación que ha

llevado a abordar dicho tema. En el tercer apartado se presentan, de manera detallada, las contribuciones de la tesis. En el cuarto apartado, se presenta la metodología de trabajo que se ha seguido durante el desarrollo de la tesis, y por último, se muestra cómo se estructura la tesis en diferentes capítulos.

Capítulo 2. Contexto Tecnológico

En este capítulo se realiza un análisis de los conceptos tecnológicos que tratan aspectos relativos a los servicios Web. Primero se presentan las arquitecturas orientadas a servicios (Service Oriented Architecture, SOA). Seguidamente, se describen las características principales y se definen los protocolos que utilizan los servicios Web (XML, WSDL, UDDI, etc.). Y finalmente, en el cuarto apartado, se presenta RBAC (Role Based Access Control) para el control de acceso a las aplicaciones.

Capítulo 3. Estado del Arte

En este capítulo se analizan los trabajos más relevantes que tratan aspectos relativos a los servicios Web, tanto a nivel de diseño como de implementación. Se describen aquellos métodos que se engloban dentro de la ingeniería Web y se comparan con la propuesta definida en esta tesis.

Capítulo 4. Un método para el diseño e implementación de servicios Web

Este capítulo provee una visión general de la propuesta de la tesis. Se presentan los modelos utilizados por OO-Method / OOWS en la *Especificación de Requisitos*, el *Modelado Conceptual* y la *Generación de Código*, así como la relación entre los modelos consumidos y generados en cada paso del *diseño e implementación de servicios Web*.

Capítulo 5. Diseño de servicios Web dirigido por modelos

En este capítulo se presentan los grupos funcionales que conforman los servicios Web de la propuesta: *Lógica de la Aplicación*, *Gestión de Usuarios*, *Recuperación de Información*, *Soporte a la Navegación* y *Soporte a la Presentación*. A continuación se analizan los modelos, se identifican las operaciones a publicar en los servicios Web indicando para cada una: qué acción se pretende llevar a cabo; cómo se obtiene cada operación y sus parámetros.

Capítulo 6. Implementación de los servicios Web

En este capítulo se presenta un proceso de generación de código para los servicios Web detectados en la etapa de modelado OO-Method / OOWS. Se define un proceso de transformación que establece las correspondencias entre los elementos de los modelos de OO-Method / OOWS y el código que utilizan los servicios Web. La propuesta aborda de forma detallada la generación de la implementación de los servicios. Los niveles de la lógica de negocio y de persistencia son generados completamente y de manera automática por el método OO-Method/OOWS. La propuesta utiliza parte de este código generado para la implementación de los servicios propuestos.

Capítulo 7. `DISWOOM`: Herramienta de soporte a la generación de servicios Web

En este capítulo se presenta la herramienta que ha sido desarrollada en el contexto del proyecto Eclipse para dar soporte a la propuesta presentada en esta tesis. También se analizan los resultados obtenidos por las herramientas que dan soporte al método OO-Method / OOWS y cómo esos resultados son utilizados por la herramienta `DISWOOM`.

Capítulo 8. Conclusiones

En este capítulo se presentan las conclusiones del trabajo y se destacan las principales contribuciones de la tesis. Además se proponen líneas de trabajo futuras relacionadas con la presente tesis.

Apéndices

Para completar el trabajo de tesis se presentan dos apéndices.

En el apéndice A, *Un caso de estudio: una aplicación de comercio electrónico*, se desarrolla un caso de estudio en el que se aplican las propuestas presentadas en esta tesis.

En el apéndice B, *Metamodelos*, se presentan los metamodelos de la *Especificación de Requisitos* y del *Modelado Conceptual*, los cuales son utilizados por la herramienta `DISWOOM`.

Capítulo 2

Contexto Tecnológico

2.1 Introducción

En este capítulo se introduce el contexto tecnológico en el que se sitúa el trabajo desarrollado en esta tesis. El contexto tecnológico comprende aquella tecnología software asociada al problema que se resuelve. Este capítulo analiza cada una de las áreas con las que está relacionado el trabajo de tesis. La combinación adecuada de técnicas presentes en cada una de las áreas, permitirá desarrollar la propuesta que se presenta.

Hoy en día existen millones de páginas y grandes cantidades de información en el World Wide Web (WWW). Con la llegada de Internet, la demanda de sitios Web comenzó a crecer rápidamente y muchas organizaciones descubrieron el potencial de la Web. La Web rápidamente se convirtió en un medio grande y poderoso para que las empresas estuvieran en contacto con clientes y proveedores, expresaran opiniones y sacaran provecho de las aplicaciones de comercio electrónico.

WWW fue creado originalmente por científicos del CERN para compartir información y documentos (ver figura 2.1, 1989). Nunca se esperó alcanzar tanta popularidad por lo que no fue diseñado para los requisitos que existen actualmente en los sitios Web. Los sitios Web necesitan ser más flexibles y poder cambiar de una forma sencilla, además necesitan proveer funcionalidad

dinámica para interactuar con otras aplicaciones existentes. El típico entorno de desarrollo Web necesita una combinación de diferentes tecnologías, herramientas y arquitecturas.

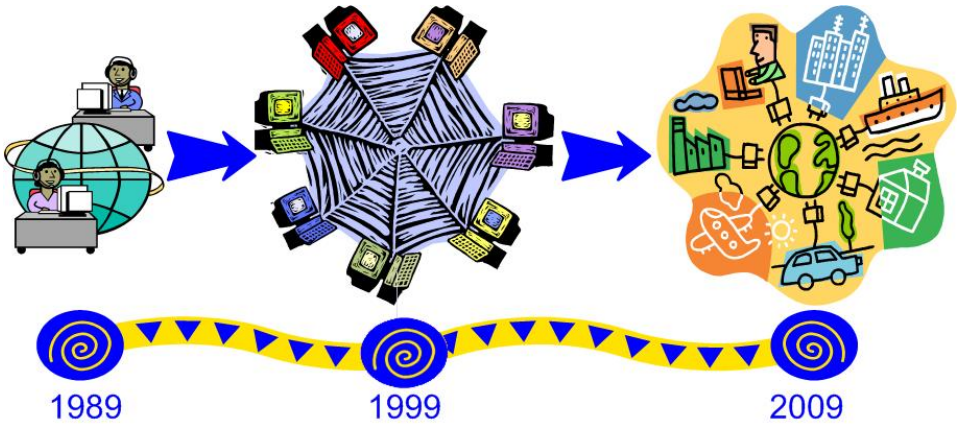

Figura 2.1: Evolución de Internet

Las oportunidades que ofrece este crecimiento de Internet han motivado un intenso trabajo en el ámbito de la investigación y en el de la industria. Flexibilidad, interconectividad, autonomía e independencia de plataforma juegan un rol fundamental en el desarrollo de software. El software se está convirtiendo cada vez más en un servicio ofrecido a los usuarios humanos o a otros elementos software, y no se ve como una aplicación aislada ejecutándose en una máquina específica para un requisito predefinido. Esta es la visión del software como «servicio» bien conceptualizado por el paradigma de computación orientada a servicios (Service Oriented Computing, SOC).

Las aplicaciones más conocidas del paradigma de SOC se pueden encontrar en la Web: las arquitecturas orientadas a servicios (Service Oriented Architecture, SOA) y los servicios Web (ver figura 2.2).

- Por un lado, SOA provee una forma de resolver problemas relacionados con la integración de aplicaciones heterogéneas en un entorno distribuido. Además, engloba todos los conceptos necesarios que se aplican en las arquitecturas de servicios, tales como orquestación, composición, seguridad, coordinación de servicios y un largo etc.

- Por otro lado, los servicios Web son un conjunto de interfaces estandarizados para la descripción, descubrimiento, invocación, composición y orquestación de elementos software independientes y poco acoplados. Los servicios Web no son parte obligatoria de SOA, pero son una implementación adecuada y la más utilizada hoy en día.

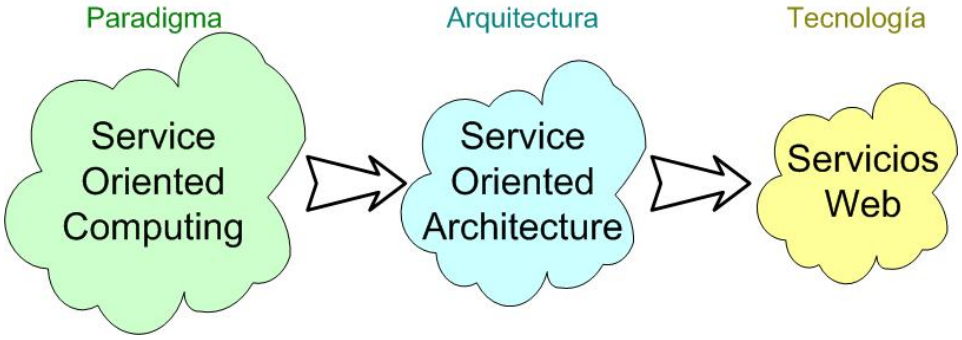

Figura 2.2: Relación entre SOC, SOA y los servicios Web

La estructura del capítulo es la siguiente: en primer lugar, se presenta una introducción a las arquitecturas orientadas a servicios (SOA). En el segundo apartado se presenta una visión general de los servicios Web, sus características, protocolos y estándares, arquitectura y tecnología que los implementa. En el tercer apartado, se analiza RBAC (la relación con el método OO-Method / OOWS se analiza en el capítulo 5, en el apartado 5.3 *Gestión de Usuarios* de la página 102). En la última parte del capítulo se presentan las conclusiones.

2.2 Arquitecturas Orientadas a Servicios

La computación orientada a servicios (SOC), es el paradigma de computación que utiliza servicios como elementos fundamentales para el desarrollo de aplicaciones y soluciones software. La arquitectura orientada a servicios (SOA), es un estilo de arquitectura cuyo objetivo es conseguir perder el acoplamiento entre servicios que interactúan.

Con el fin de evaluar el momento que vive la empresa hacia SOA y lo que supone para el negocio, expertos de firmas proveedoras, desarrolladoras

e integradoras (BEA Systems, Gartner, Borland, HP, Coritel, Rational Edge o Cape Clear entre otros) han analizado las implicaciones de este concepto. Según estas empresas, SOA presenta la agilidad, flexibilidad y el ahorro de costes como premisas de un modelo que permite unir TI con los requerimientos del negocio. Para ellas, lo importante es crear aplicaciones que estén pensadas desde su inicio para interactuar e integrarse con otras aplicaciones.

Las empresas están realmente interesadas en que SOA evolucione y se integre totalmente en el proceso de desarrollo de nuevas soluciones.

2.2.1 Definición

Como se ha visto, las empresas están interesadas en utilizar SOA como una prioridad a la hora de implementar una solución. Existen múltiples definiciones de SOA ([37, 38, 39]), pero la más aceptada es la proporcionada por W3C [40]: "SOA es un conjunto de componentes que pueden ser invocados, cuyas descripciones de interfaces se pueden publicar y descubrir".

Una arquitectura orientada a servicios tiene varios elementos fundamentales que deben aparecer. El principal concepto es «servicio» y las colaboraciones principales son «publicar», «descubrir/buscar» e «interactuar». A través de este capítulo se va a conocer cada uno de estos conceptos.

2.2.2 Roles

Existen tres roles dentro de SOA (ver figura 2.3):

- *Proveedor de servicios*: implementa el servicio y lo hace accesible en Internet.
- *Consumidor de servicios*: interactúa con un proveedor de servicios mediante mensajes XML. Tradicionalmente se le llama «cliente». Puede ser una aplicación final u otro servicio.
- *Repositorio de servicios*: provee un lugar donde los desarrolladores pueden publicar nuevos servicios y buscar otros existentes.

Figura 2.3: Roles de SOA

Cada entidad en SOA, puede jugar uno o más de los tres roles de proveedor, consumidor o repositorio de servicios. En la figura 2.3, se muestran los tres tipos de colaboración entre los roles:

- *Publicar servicio*: un proveedor de servicios publica un servicio, haciéndolo disponible a los consumidores a través de un repositorio de servicios.
- *Buscar servicio*: los consumidores de servicios buscan y localizan los servicios en un repositorio de servicios.
- *Interactuar*: es la comunicación entre el consumidor y los servicios del proveedor. El consumidor realiza peticiones al servicio a través de los protocolos que indica la información del servicio que tiene el repositorio.

2.2.3 Diseño

El diseño de SOA impone una serie de requisitos que deben considerarse en el desarrollo de aplicaciones y que afectan a los métodos de producción de software. Estos requisitos son:

- *Vista Lógica y Pública*: Los servicios son una abstracción (vista lógica) de los programas, bases de datos, procesos de negocio, etc., definidos en términos de lo qué hace el sistema (llevando a cabo una operación de negocio) y de lo que se hace público (se publican descripciones de aquellos detalles que se exponen públicamente y son relevantes para el uso del servicio).

- *Orientación a Mensajes y Acoplamiento Débil*: El servicio se define en términos de los mensajes (síncronos y asíncronos) intercambiados entre proveedores y solicitantes. SOA promueve soluciones no orientadas a la conexión y basadas en mensajes, lo que proporciona un acoplamiento débil.

- *Orientación a la Descripción*: Un servicio se describe con metadatos procesables. La descripción da soporte a la naturaleza pública de SOA: solo se incluyen en la descripción, aquellos detalles que se exponen públicamente y son importantes para el uso del servicio. La semántica de un servicio debe documentarse, directa o indirectamente, en su descripción.

- *Granularidad*: Los servicios tienden a usar un número de operaciones. La complejidad de estas operaciones depende de la granularidad de las mismas (fina o gruesa). En la página 86 del capítulo 5, se presentan más detalles sobre la granularidad adoptada en el desarrollo de esta tesis.

- *Neutral a la Plataforma*: los mensajes se envían en un formato estándar y neutral a la plataforma, distribuido a través de las interfaces (XML).

2.3 Servicios Web

Como se ha comentado en el apartado anterior, una aplicación SOA está formada por un conjunto de servicios que encapsulan los procesos de negocio. Los servicios realizan funciones que pueden ir desde la más simple respuesta hasta el más complicado proceso de negocio. Los servicios permiten a las organizaciones, exponer su funcionalidad sobre Internet usando lenguajes y protocolos estándares (basados en XML), y son implementados mediante interfaces autodescriptivas, basadas en estándares abiertos.

Las instancias más conocidas de servicios, son los servicios Web. Estos proporcionan la plataforma tecnológica ideal para conseguir la completa integración de los procesos de negocio de una organización con diferentes organizaciones. Los servicios Web prometen ser el mecanismo adecuado para la implementación de SOA en sistemas integrados y distribuidos. En este apartado se presenta cómo se define un servicio Web, sus características, arquitectura y los protocolos y estándares que lo definen.

2.3.1 Definición

Existen múltiples definiciones sobre lo que son los servicios Web, lo que muestra su complejidad a la hora de dar una adecuada definición que englobe todo lo que son e implican. Pero la definición más aceptada es la proporcionada por W3C.

W3C [41] los define como "un conjunto de aplicaciones o de tecnologías con capacidad para interoperar en la Web. Estas aplicaciones o tecnologías intercambian datos entre sí con el objetivo de ofrecer unos servicios. Los proveedores ofrecen sus servicios como procedimientos remotos y los usuarios solicitan un servicio llamando a estos procedimientos a través de la Web".

2.3.2 Características

Si se quiere dar una definición basada en las características que ha de satisfacer un servicio Web, se puede definir como servicio Web un proceso que:

- *Expone y describe sus características*: un servicio Web define sus funcionalidades y propiedades de tal forma que las demás aplicaciones pueden tener un conocimiento completo del tipo de servicio que se está ofreciendo.

- *Es localizable en la red*: se puede registrar el servicio Web en un repositorio, de tal forma que las aplicaciones remotas lo puedan localizar y acceder con sencillez.

- *Se puede invocar*: una vez que el servicio Web requerido ha sido localizado y analizado para controlar que satisfaga las necesidades, la aplicación remota lo puede invocar.
- *Devuelve una respuesta*: cuando el servicio ha sido ofrecido a la aplicación remota, se le devuelve el resultado.

Además de las características anteriores, debe de cumplir las características generales que tienen los servicios [2]:

- *Tecnológicamente neutrales*: deben ser invocables a través de cualquier tecnología. Esto implica que los mecanismos de invocación (protocolos, mecanismos de descripción y descubrimiento) deben cumplir estándares.
- *Sin acoplamiento*: no deben requerir el conocimiento de estructuras internas ni por parte del cliente ni del proveedor.

2.3.3 Escenarios

A continuación se presentan los tres escenarios principales en el desarrollo de servicios Web [2, 42]:

1. *Exponer la funcionalidad existente de una aplicación a través de un servicio Web* (ver figura 2.4). Este escenario es el más utilizado actualmente. Muchas empresas han invertido mucho tiempo y dinero publicando la funcionalidad que ya tenían implementada a través de los servicios Web. Para ello, implementan una capa SOAP/WSDL/UDDI por encima de sus aplicaciones ya existentes. Pero la clave de los servicios Web no es integrar aplicaciones, sino crear aplicaciones que sean flexibles, reutilizables e interoperables. Con este tipo de escenarios se pierden características importantes de los servicios Web.

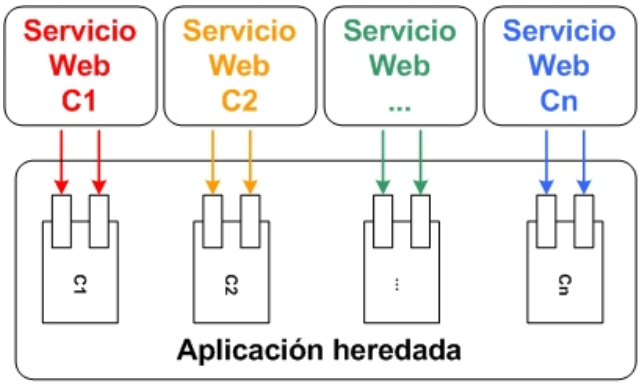

Figura 2.4: SOA: Escenario 1

2. *Implementar las aplicaciones como nuevos servicios Web* (ver figura 2.5). Las organizaciones ofrecen sus productos y servicios a través de Internet para encontrar clientes y socios, e integrar sus aplicaciones con otras. Para ello diseñan e implementan nuevos servicios Web que les permiten mantener intactos todos los beneficios que estos les ofrecen. El trabajo de tesis se centra en este tipo de escenarios.

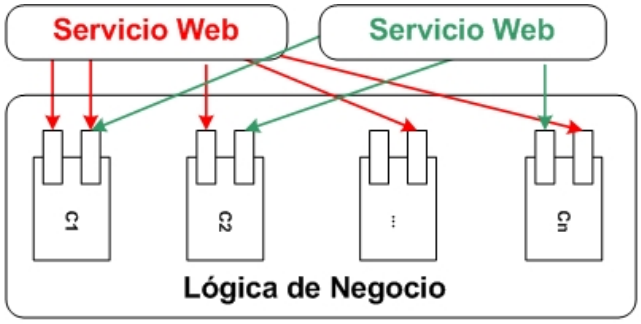

Figura 2.5: SOA: Escenario 2

3. *Integrar servicios Web de otros vendedores o socios de negocios* (ver figura 2.6). Este escenario es muy interesante, pero el problema a tratar en este trabajo de tesis se centra en los servicios propios. Existen diversos

trabajos relacionados con este tipo de escenarios [43, 44, 45] dentro del grupo de investigación.

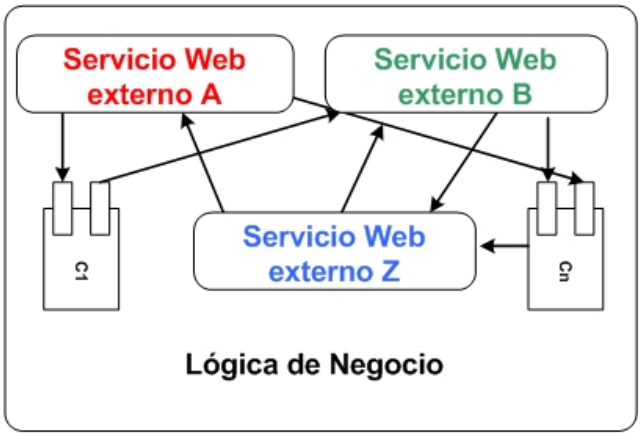

Figura 2.6: SOA: Escenario 3

El escenario 2 es el principal escenario de la tesis, pero el trabajo realizado también puede ser útil para los otros dos escenarios. Por ejemplo, para el escenario 1 se puede reutilizar el diseño de servicios Web para crear la capa que expone la funcionalidad existente de una aplicación. En cuanto al escenario 3, se puede adaptar el servicio Web de soporte a la presentación para integrar servicios Web de terceros.

2.3.4 Arquitectura

La infraestructura de comunicación de los servicios Web está soportada por arquitecturas orientadas a servicios, comentadas en el apartado anterior. En la figura 2.7 se muestra la arquitectura presentada en la figura 2.3, indicando las distintas interacciones y protocolos usados para los servicios Web:

1. Un servicio Web, publica su definición WSDL en un repositorio de servicios UDDI.
2. El cliente, busca la definición del servicio en el registro.

3. El registro, usa la información de la definición WSDL para enviar peticiones al servicio vía SOAP.

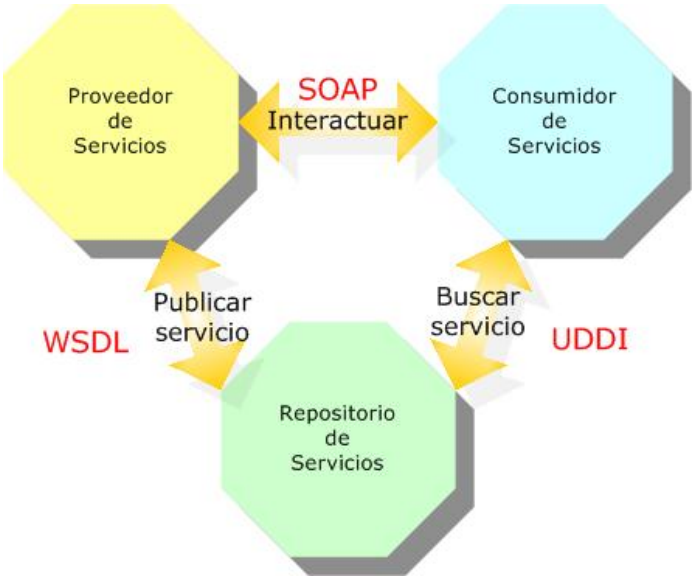

Figura 2.7: Interacción en una Arquitectura Orientada a servicios Web

En todo este proceso intervienen una serie de protocolos y estándares que hacen posible esta circulación de información. A continuación se presentan los protocolos y estándares más utilizados en los servicios Web.

2.3.5 Protocolos y estándares

Bajo los servicios Web subyacen una serie de protocolos y estándares que ofrecen interoperabilidad, seguridad y confiabilidad a todos los aspectos que rodean a los servicios. Estos protocolos y estándares tratan de describir y definir todo lo referente a los servicios Web, desde cómo se describen o localizan, hasta cómo se representa toda la información, pasando por la forma mediante la cual se deben comunicar.

Desde el punto de vista técnico, los estándares que actualmente han conseguido una considerable importancia en el paradigma de los servicios Web,

por lo que se refiere a publicación de los servicios e interconectividad de los mismos, son sobre todo el protocolo SOAP, el lenguaje WSDL y la interfaz UDDI, que forma la pila de protocolos de los servicios Web.

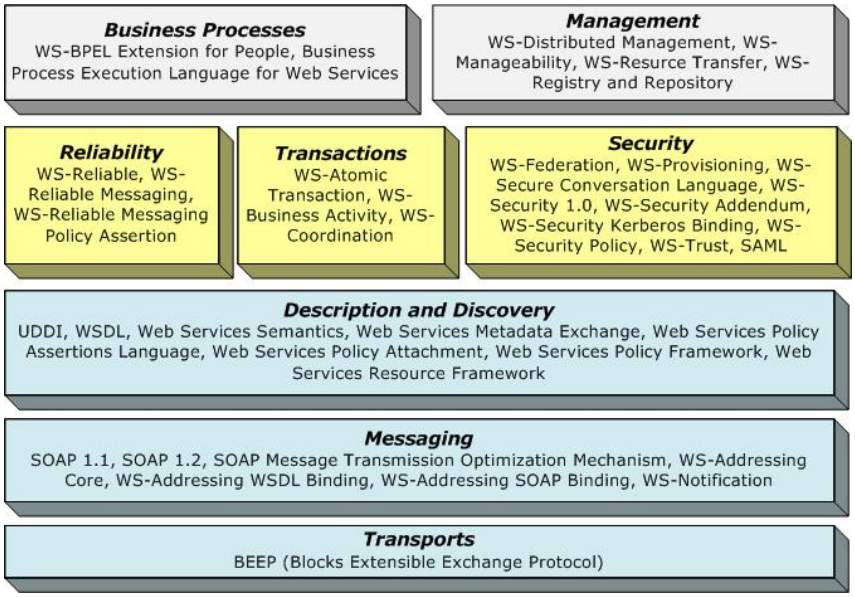

Figura 2.8: Pila de protocolos de los servicios Web

La pila de protocolos de los servicios Web (ver figura 2.8 [1]), es un conjunto de protocolos utilizados para definir, descubrir e implementar servicios Web. El núcleo de la pila de protocolos (ver figura 2.9) consiste en cuatro capas [46]:

- *Servicio de transporte*: esta capa es responsable del transporte de mensajes entre aplicaciones. Actualmente, esto incluye HTTP, SMTP, FTP, etc.
- *Codificación de datos y mensajes*: esta capa es la responsable de codificar los mensajes en un formato común XML, de tal forma que los mensajes puedan ser entendidos. Actualmente, esto incluye XML-RPC y SOAP.
- *Descripción de servicios*: esta capa es la responsable de describir la interfaz pública de un servicio Web específico. Actualmente, para la descripción de servicios se utiliza WSDL.

[1] https://www.ibm.com/developerworks/webservices/standards/

- *Descubrimiento de servicios*: esta capa es responsable de centralizar los servicios en un registro común y proveer una forma fácil de publicar/encontrar funcionalidad. Actualmente, para el descubrimiento de servicios se utiliza UDDI.

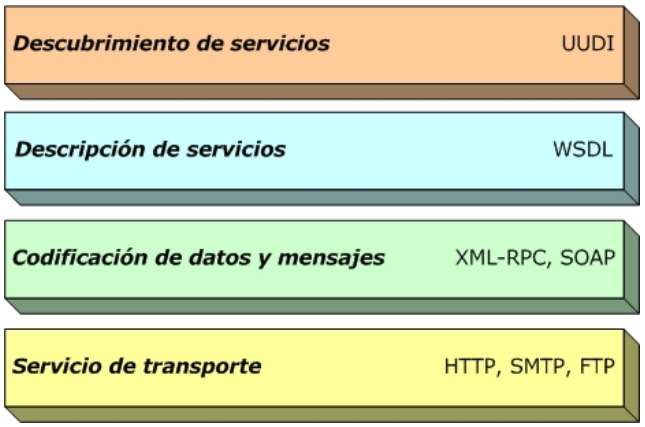

Figura 2.9: Pila básica de protocolos de los servicios Web

XML

XML (eXtensible Markup Languageble Markup Language) [5], se utiliza para normalizar el intercambio de datos entre participantes proporcionando un medio excelente para codificar y formatear los datos. XML es similar a HTML, con elementos, atributos y valores.

Los elementos y atributos de XML, definen tipos y estructuras de información para los datos que llevan, incluyendo la capacidad de modelar datos y estructuras específicas a un dominio de sistema. Un aspecto fundamental de los servicios Web es transformar un XML genérico de datos en una aplicación, o en una representación de dominio específico de datos.

La sintaxis de XML usada en las tecnologías de los servicios Web específica cómo se representan los datos, define cómo y con qué calidad se transmiten los datos y los detalles de cómo se publican y descubren los servicios.

SOAP

SOAP (Simple Object Access Protocol) [6], proporciona un modo abierto y extensible para que las aplicaciones se comuniquen a través de la Web usando mensajes basados en XML, con independencia de sistemas operativos, modelos de objetos o lenguajes de programación.

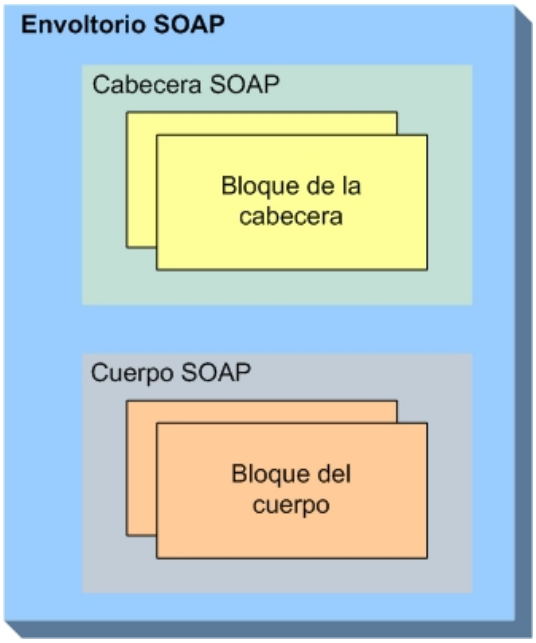

Figura 2.10: Estructura de un mensaje SOAP

Facilita la comunicación universal, definiendo un formato de mensajes simple y extensible en XML estándar y proporcionando además, un modo para enviar esos mensajes sobre HTTP.

SOAP intercambia información mediante mensajes. Los mensajes se utilizan como envoltorios que la aplicación utiliza para guardar la información que quiere enviar. Cada envoltorio contiene dos partes (ver figura 2.10): (1) una cabecera (opcional) y (2) un cuerpo (obligatorio). La cabecera y el cuerpo pueden tener múltiples subpartes en forma de bloques.

WSDL

WSDL (Web Services Description Language) [7] es un lenguaje XML para la descripción de servicios Web, permitiendo separar la descripción de la funcionalidad abstracta ofrecida por un servicio, de los detalles concretos de la descripción del servicio.

El documento WSDL de un servicio, proporciona dos piezas de información básicas (ver figura 2.11): (1) una parte o interfaz abstracta (independiente de la aplicación), y (2) una parte concreta que define los enlaces a protocolos e información de los puntos finales de acceso al servicio.

La **parte abstracta** está compuesta por:

- *Type*: es un contenedor de definiciones de tipos de datos, usado por las operaciones (por defecto XML schema).
- *Message*: es una unidad de comunicación representando un intercambio de datos en una única transmisión lógica.
- *Operation*: descripción abstracta de una operación soportada por el servicio. Define un intercambio simple de mensajes.
- *Port Type*: colección lógica de operaciones soportadas por uno o más puntos finales.

La **parte concreta** está compuesta por:

- *Binding*: especifica la codificación de los mensajes y los enlaces a protocolos de todas las operaciones y mensajes definida en un port type.
- *Port*: especifica en qué dirección (URI) se puede acceder a la implementación del port type. Definen un punto final (lugar de la red), donde está el servicio.
- *Service*: define una agrupación de ports.

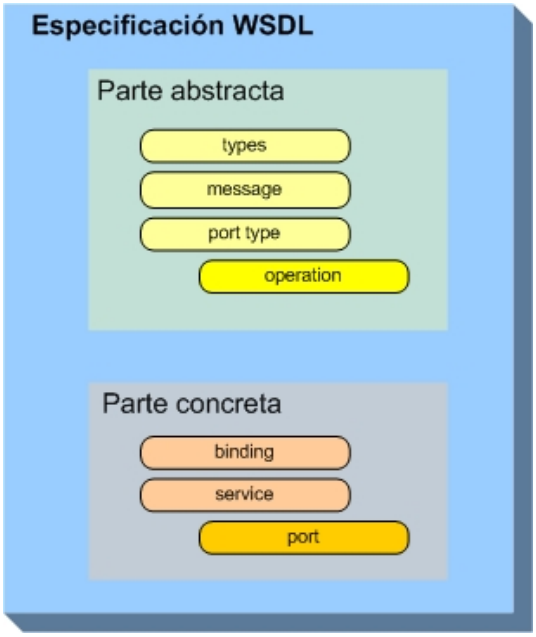

Figura 2.11: Especificación de un documento WSDL

De esta forma, WSDL se utiliza para describir un servicio Web en términos de los mensajes que acepta y genera, actúa como contrato entre un consumidor (cliente) y dicho servicio. WSDL puede describir puntos finales y sus operaciones, sin especificar el formato de los mensajes o protocolos de red a los cuales el punto final esta ligado.

UDDI

UDDI (Universal Description, Discovery, and Integration) [8], es un registro público, diseñado para almacenar de forma estructurada información sobre empresas y los servicios que éstas ofrecen. A través de UDDI, se puede publicar y descubrir información de una empresa y de sus servicios.

La especificación UDDI tiene dos objetivos esenciales:

1. Ser un soporte a los desarrolladores para encontrar información sobre

servicios Web y poder construir clientes.

2. Facilitar el enlace dinámico de servicios Web, permitiendo consultar referencias y acceder a servicios de interés.

2.4 Role Based Access Control

El proceso de desarrollo de este estándar fue iniciado por el *National Institute of Standards and Technology (NIST)*, debido a la necesidad existente entre los compradores de sistemas de información de una definición clara y precisa, cuyas características de control de acceso no fueran sufriendo cambios. Durante muchos años, los vendedores han estado implementando características de control de acceso basado en roles en sus sistemas, sin un acuerdo general de cuáles eran las características que debe tener todo sistema de control de acceso basado en roles (RBAC) [47]. Esta falta de consenso de un modelo, causó incertidumbre y confusión sobre la utilidad y el significado de RBAC.

El estándar *Role Based Access Control (RBAC)* [48], es un estándar para el control de acceso basado en roles que apareció para resolver esta situación. RBAC está formado por dos partes principales: el *modelo de referencia* y su *especificación funcional*.

El *Modelo de Referencia de RBAC*, define conjuntos de elementos RBAC básicos (por ejemplo, usuarios, roles, permisos, operaciones y objetos) y relaciones, como tipos y funciones que están incluidos en este estándar. El modelo de referencia de RBAC se define en términos de cuatro modelos: *Core RBAC* (CRBAC), *Hierarchical RBAC*, *Static Separation of Duty Relations* y *Dynamic Separation of Duty Relations*. En el trabajo de tesis sólo se ha utilizado el modelo de Core RBAC, por lo que es el único que se comenta a continuación.

La *Especificación Funcional de RBAC*, define las características que se requieren en un sistema RBAC. Define operaciones para crear, borrar y mantener los elementos RBAC y sus relaciones; funciones que permiten llevar a cabo operaciones de consulta sobre elementos RBAC y sus relaciones; y funciones para la creación de sesiones de usuarios que incluyen la activación/desactivación de roles, etc.

2.4.1 Core RBAC

El conjunto de elementos y relaciones del modelo de Core RBAC (CRBAC) se definen en la figura 2.12. CRBAC incluye un conjunto de cinco elementos básicos de datos llamados `usuarios`, `roles`, `objetos`, `operaciones` y `permisos`. El modelo RBAC completo se define fundamentalmente en términos de usuarios individuales que se asignan a roles y permisos que se asignan a roles.

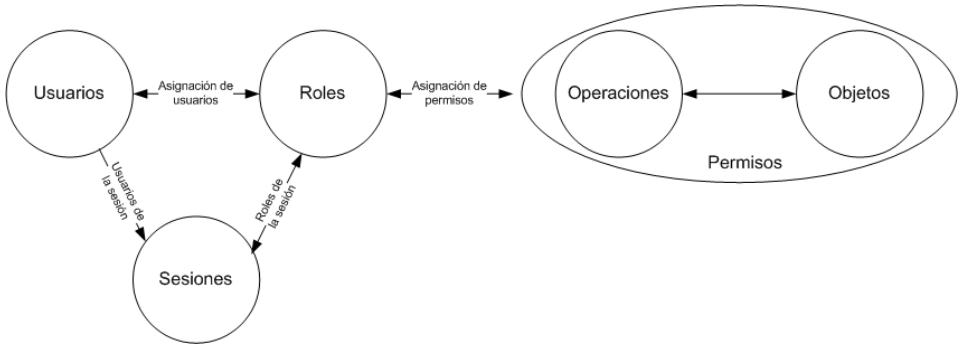

Figura 2.12: Elementos y relaciones del modelo de Core RBAC

El significado de los elementos básicos del modelo CRBAC son:

- `Objeto`: puede ser cualquier recurso del sistema que necesite control de acceso, como puede ser ficheros, impresoras, bases de datos, etc.
- `Operación`: funcionalidad ejecutable por el usuario.
- `Permiso`: consentimiento de llevar a cabo una operación sobre alguno de los objetos RBAC.
- `Rol`: función dentro del contexto de una organización.
- `Usuario`: persona que puede acceder al sistema.

Los tipos de operaciones y objetos que controla RBAC dependen del tipo de sistema en el cuál están implementados. La figura 2.12 muestra gráficamente, además de los elementos básicos que componen RBAC, las relaciones existentes entre los distintos elementos.

A continuación se comentan aquellas operaciones necesarias para administrar correctamente los elementos que se encuentran en CRBAC.

- `AddUser`: esta operación crea un nuevo usuario RBAC. La operación solo es válida si el nuevo usuario no existe todavía como miembro del conjunto de `usuarios`.
- `DeleteUser`: esta operación elimina un usuario existente de la base de datos de RBAC. La operación es válida si y solo si el usuario que va a ser borrado es miembro del conjunto de `usuarios`. Los datos de usuarios y roles que tenía asignados se actualizan. Es una decisión de implementación cómo proceder con la sesión del usuario a ser borrado. El sistema RBAC puede esperar a que el usuario termine su sesión normalmente o puede forzar su finalización.
- `AddRole`: esta operación crea un nuevo rol. La operación es válida si y solo si el nuevo rol no existe todavía como miembro del conjunto de `roles`. Inicialmente, el nuevo rol no tendrá ningún usuario ni ningún permiso asignado.
- `DeleteRole`: esta operación elimina un rol existente de la base de datos de RBAC. La operación es válida si y solo si el rol que va a ser borrado es miembro del conjunto de `roles`. Es una decisión de implementación cómo proceder con las sesiones que tienen activo el rol que va a ser borrado. El sistema RBAC puede esperar a que la sesión termine normalmente o puede eliminar el rol de las sesiones mientras se permite que las sesiones continúen.
- `AssignUser`: esta operación asigna un usuario a un rol. La operación es válida si y solo si el usuario existe en el conjunto de `usuarios`, el rol existe en el conjunto de `roles` y el usuario no ha sido todavía asignado al rol.
- `DeassignUser`: esta operación elimina la asignación de un usuario de un rol. La operación es válida si y solo si el usuario existe en el conjunto de `usuarios`, el rol existe en el conjunto de `roles` y el usuario ha sido asignado al rol.
- `GrantPermission`: esta operación asigna a un rol el permiso de llevar a cabo una operación sobre un objeto. La operación es válida si y solo si

el par (operación, objeto) representa un permiso, y el rol es un miembro del conjunto de `roles`.

- `RevokePermission`: esta operación cancela el permiso de llevar a cabo una operación sobre un objeto del conjunto de permisos asignados a un rol. La operación es válida si y solo si el par (operación, rol) representa un permiso, el rol existe en el conjunto de `usuarios` y el permiso está asignado al rol.

Además de las operaciones de administración de usuarios, roles y permisos, RBAC proporciona especificación funcional para la administración de las sesiones.

- `CreateSession(user, session)`: esta operación crea una nueva sesión con el usuario y un grupo de roles activos. La operación es válida si y solo si el usuario es miembro del conjunto de `usuarios` y el conjunto de roles activos es un subconjunto de roles asignados al usuario.
- `DeleteSession(user, session)`: esta operación elimina la sesión del usuario. La operación es válida si y solo si la sesión es miembro del conjunto de `sesiones`, el usuario existe en el conjunto de `usuarios` y el usuario es el propietario de la sesión.

Core RBAC proporciona también operaciones de consulta sobre los elementos de RBAC y sus relaciones.

- `AssignedUsers`: esta operación devuelve el conjunto de usuarios asignados a un rol. La operación es válida si y solo si el rol es miembro del conjunto de `roles`.
- `AssignedRoles`: esta operación devuelve el conjunto de roles asignados a un usuario. La operación es válida si y solo si el usuario es miembro del conjunto de `usuarios`.

Finalmente, Core RBAC proporciona funciones avanzadas para la consulta de los permisos.

- `RolePermissions`: esta operación devuelve el conjunto de permisos (operación, objeto) de un rol. La operación es válida si y solo si el rol es miembro del conjunto de `roles`.

2.5 Conclusiones

En este capítulo se han presentado las características de SOA, de los servicios Web y de RBAC, elementos fundamentales para el desarrollo de esta tesis.

Los servicios Web facilitan el acceso a la funcionalidad de las aplicaciones a través de Internet, permitiendo ir más allá de los componentes software. Aportan grandes ventajas como el acceso a servicios desde cualquier punto de la red, simplifican el acceso a la funcionalidad de distintas aplicaciones empresariales. Además, suponen un avance respecto a los componentes porque permiten realizar alquiler de servicios externos frente a la compra o desarrollo de componentes. Los servicios Web se están convirtiendo en la tecnología que está facilitando el desarrollo de aplicaciones Web de comercio electrónico, y pronto convertirán la Web en un marco para el desarrollo de aplicaciones distribuidas, extendiéndose a todos los dominios de aplicación.

En general, SOA y los servicios Web son apropiados para aplicaciones:

- Que deben operar a través de Internet, donde la fiabilidad y la velocidad no se pueden garantizar.
- Donde no existe la posibilidad de gestionar la instalación de forma que todos los clientes y proveedores se actualicen a la vez.
- Donde los componentes se ejecuten en distintas plataformas y distintos productos.
- Donde una aplicación existente necesite exponerse para ser usada a través de la red y pueda decorarse como un servicio Web.

Capítulo 3

Estado del Arte

Este capítulo ofrece una visión general del estado del arte en el área de interés de la tesis. Este área está formada por el área de Ingeniería Web. El capítulo comienza con una visión general de las propuestas más conocidas en el ámbito de la Ingeniería Web. En particular, esta visión general se centra en las soluciones tomadas para resolver el diseño e implementación de servicios Web. El siguiente apartado presenta un análisis comparativo de las principales propuestas presentadas en el apartado anterior. Por último, se cierra el capítulo con un análisis de las propuestas presentadas.

3.1 Introducción

El diseño de servicios Web es la actividad, por medio de la cual, las operaciones de los servicios son obtenidas. Estas operaciones deben tener una granularidad adecuada, ni muy finas ni muy gruesas. La granularidad en un servicio Web es importante puesto que un servicio Web de grano fino conlleva muchas llamadas ligeras (con menos datos) para llevar a cabo una acción mientras que un servicio Web de grano grueso conlleva menos llamadas pero más pesadas (con más datos)[1].

[1]En el capítulo 5 se comenta la importancia de la granularidad

No sólo es importante la obtención de las operaciones que conforman los servicios Web, sino proporcionar una guía que ayude a los diseñadores y modeladores a identificar correctamente los servicios Web a publicar para sus sistemas.

El método propuesto en esta tesis se centra en proporcionar servicios Web de granularidad óptima, proporcionando una guía y una herramienta que automatiza el diseño e implementación de los servicios Web. A continuación, se introducen las aproximaciones más relevantes de los métodos de Ingeniería Web.

3.2 Ingeniería Web

A finales de los años 90 surgió una nueva disciplina conocida como Ingeniería Web (*Web Engineering*) [49]. Esta nueva disciplina se define en [9] como "*El establecimiento y uso de principios de ingeniería de gestión, así como de enfoques sistemáticos y disciplinados para el desarrollo, instalación y mantenimiento exitoso de sistemas y aplicaciones de un alta calidad basadas en la Web*". Diferentes autores [50, 51, 52] han dado soporte a esta nueva ingeniería para cubrir las necesidades introducidas por los sistemas basados en la Web. Como resultado de los estudios realizados en este área, se definieron un conjunto de métodos, modelos y técnicas.

De este conjunto de métodos, las propuestas más conocidas en el ámbito de la Ingeniería Web y que se analizan en esta tesis son Hera [53], OO-H [54], OOHDM [55], UWE [56], WebML [57] y WSDM [58]. En particular, este análisis se centra en las soluciones dadas por estos métodos para el diseño e implementación de servicios Web. Se presentan ordenados alfabéticamente, y para casa uno de estos métodos se cubren los siguientes puntos:

- El apartado comienza con una breve descripción sobre cada método.
- A continuación se comentan las etapas del método.
- Se realiza un estudio de la adaptación del método para el diseño e implementación de los servicios Web y si se proponen guías para la obtención de las operaciones desde los modelos conceptuales del método.

3.2.1 Hera

Hera [53, 59] es una metodología de diseño dirigida por modelos y un framework de especificación que se centra en el desarrollo de WIS (Web-based Information Systems) dependientes de contexto o personalizados.

La herramienta que da soporte a Hera se llama `Hera CASE tool`. Esta herramienta utiliza RDF/XML para la serialización de los distintos modelos y XSLT para la transformación entre los modelos y sus instancias.

Proceso de desarrollo

Hera se basa en la separación de conceptos y distingue tres etapas de diseño:

- *Diseño conceptual y de integración.* La principal salida de esta fase son los modelos conceptuales y de integración (CM, IM). CM da una visión semántica del repositorio de datos, mientras que IM especifica los enlaces semánticos de los conceptos de orígenes particulares a conceptos en CM.
- *Diseño de la aplicación y de la adaptación.* En esta fase el diseñador crea un modelo de la aplicación (AM), y un conjunto de modelos para la adaptación. AM se construye por encima de CM y describe toda la estructura de presentaciones generadas incluyendo la navegación. La adaptación de la presentación generada se basa en propiedades estáticas o dinámicas basadas en un modelo de usuario.
- *Diseño de la presentación.* En esta fase el diseñador especifica la capa de presentación en unidades de presentación.

Adaptación a los servicios Web

Actualmente, existe una extensión de Hera [28] que propone la transformación de componentes Hera a dos tipos de servicios Web (ver figura 3.1[2]): un servicio

[2]Figura extraída del artículo [28]

Web que provee datos y un servicio Web que conoce como se presentan estos datos.

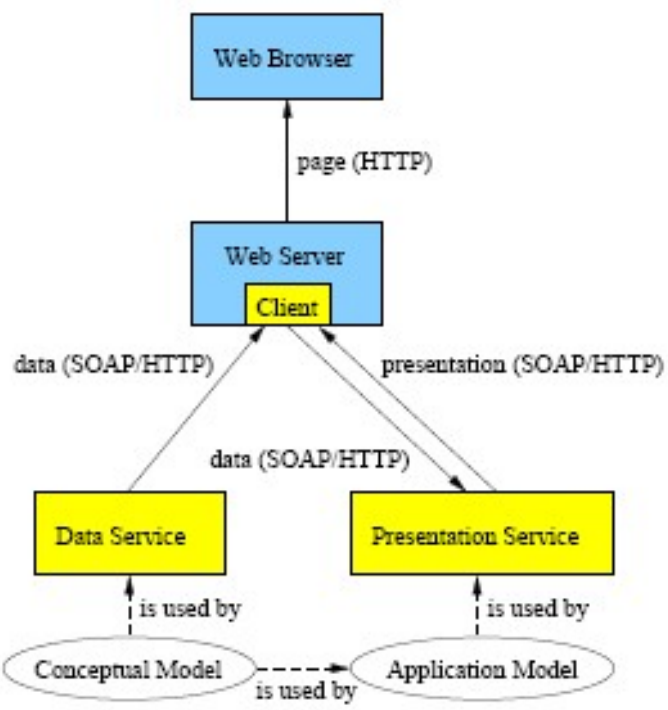

Figura 3.1: Arquitectura Orientada a Servicios de Hera

La propuesta presentada por Hera no aborda cómo se implementan estos servicios Web ni propone un método para derivar servicios Web a partir de los modelos Hera.

Como trabajos futuros en [28] se quiere extender los servicios Web con nuevos tipos como servicios de recuperación de consultas, de integración de datos o para generar presentaciones dinámicas.

3.2.2 OO-H: Object-Oriented Hypermedia Method

El método OO-H (Object-Oriented Hypermedia Method) [54, 60] es un método que provee un framework basado en estándares para capturar las propiedades relevantes que participan en el modelado e implementación de los interfaces para aplicaciones Web. OO-H comenzó como una extensión de OO-Method [61], pero en publicaciones recientes los autores desasocian OO-H de OO-Method y se centran en la personalización de sitios Web [62]. La metodología tiene dos vistas:

- La *vista navegacional* extiende el diagrama de clases con propiedades de la navegación hipermedia, y está definida en NADs (Navigational Access Diagrams).
- La *vista de presentación* usa diferentes elementos relacionados con la apariencia del interfaz y con el comportamiento para modelar un número de estructuras de plantillas interconectadas expresadas en XML. Esta vista está definida en APDs (Abstract Presentation Diagrams).

La herramienta que da soporte al método OO-H es `VisualWADE`[3] (Web Application Development Environment). VisualWADE provee un entorno que soporta todos los aspectos metodológicos de OO-H, simplifica el diseño y la implementación de los sistemas de información basados en Web desde una perspectiva orientada a objetos.

Este software, basado en estándares para el análisis y diseño de sistemas de información (UML, OCL, XML), facilita: un entorno para modelar interfaces de usuarios personalizadas e independientes de lenguajes para Web; compiladores de modelos para la generación automática de interfaces operativos en varios lenguajes: HTML/CSS, XML, ASP, JSP, PHP, etc.; conexión entre los interfaces generados y sistemas legados y/o servicios Web.

Proceso de desarrollo

OO-H propone un proceso de desarrollo con cuatro etapas:

[3] http://www.visualwade.com

- En la etapa de *análisis de requisitos* se especifican los requisitos de cada tipo de usuario.
- La segunda etapa es la de *ingeniería*, en la que se realizan actividades relacionadas con el análisis y diseño del producto software. En particular, estas actividades son cinco: análisis del dominio, análisis navegacional, diseño del dominio, diseño de la navegación y diseño de la presentación.
- La siguiente etapa es la *construcción y adaptación* y constituye la implementación de la aplicación Web.
- La última etapa es la de *evaluación del cliente*, donde el cliente evalúa la aplicación Web desarrollada.

Adaptación a los servicios Web

El método OO-H no proporciona herramientas para el diseño de servicios Web en `Visual Wade`. Además, no se han encontrado publicaciones que presenten guías para identificar servicios Web a partir de sus modelos conceptuales.

A pesar de que el método OO-H no permite el diseño de servicios Web, `Visual Wade` si que permite generar la lógica de la aplicación por medio de servicios Web. La desventaja existente, es que al no haber un diseño previo, todas las operaciones modeladas serán implementadas como un servicio Web.

3.2.3 OOHDM: Object-Oriented Hypermedia Design Method

OOHDM (Object-Oriented Hypermedia Design Method) [55, 63] es un método de diseño orientado a objetos (OO) que usa conceptos y técnicas OO para construir aplicaciones hipermedia. Este método se basa en las ideas propuestas por HDM [64].

La herramienta que da soporte a OOHDM es HyperDE [4]. Esta herramienta está basada en el método SHDM (Semantic Hypermedia Design Method)

[4] http://www.tecweb.inf.puc-rio.br:8000/hyperde

[65], una variante de OOHDM, que permite describir semánticamente datos y metadatos.

Proceso de desarrollo

Las etapas que sigue el método OOHDM son cinco:

- En la etapa de *captura de requisitos* identifica los usuario que van a interactuar con la aplicación Web así como las necesidades del usuario que debe cubrir la aplicación Web. Esta etapa fue incluida tras la definición de algunas extensiones al método [66].
- El *diseño conceptual* consiste en la definición del esquema conceptual donde se definen los aspectos estáticos del sistema.
- La etapa de *diseño navegacional* se define un diagrama de clases navegacional y un diagrama de estructura navegacional. El primero representa la parte estática del sistema navegacional. El segundo extiende el diagrama de clases navegacional incluyendo estructuras de accesos y contextos navegacionales.
- El *diseño de la interfaz abstracta* consiste en la descripción de la interfaz del usuario de una forma abstracta. Este diseño se realiza mediante la utilización de ADVs (Abstract Data Views) [67].
- La etapa de *implementación* de la aplicación Web se basa en los modelos anteriores transformándolos en objetos de implementación.

Adaptación a los servicios Web

OOHDM [29] representa las acciones de los servicios Web propios como un interfaz con el estereotipo «action service interface» en el esquema conceptual (a nivel de modelado), el cual no solo representa el interfaz del servicio sino también el componente que lo implementa.

En la figura 3.2[5] se observa cómo se modelan los servicios Web en OOHDM.

[5]Figura extraída del artículo [29]

En la parte izquierda de la figura se encuentra el esquema conceptual de OOHDM con el servicio Web `HotelService` (etiquetado como *action service interface*) que lo implementa el componente `HotelServiceComponent`. En la parte derecha de la figura se encuentra el esquema navegacional de OOHDM con la asociación directa desde el `HotelInfoNode` al `HotelService`.

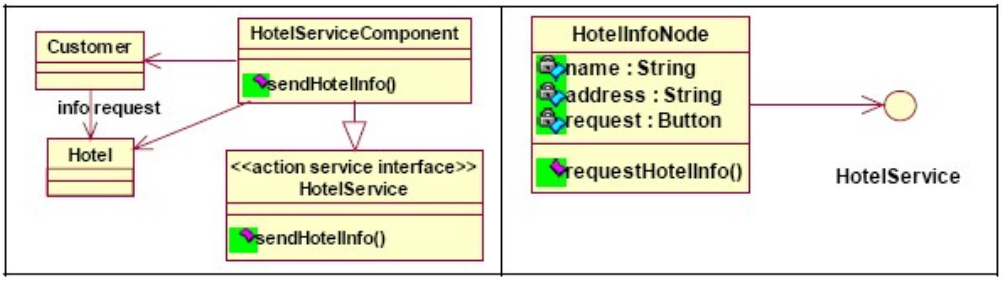

Figura 3.2: Modelado de servicios Web en OOHDM

OOHDM provee mecanismos de abstracción para incluir llamadas a servicios Web en la definición de sus modelos conceptuales. Sin embargo, la propuesta presentada no provee soporte metodológico para el desarrollo de servicios Web a partir de sus modelos conceptuales ni se ha encontrado ninguna publicación donde presenten cómo se genera la implementación de los servicios Web modelados.

Como se comenta en [63] tanto SHDM como la herramienta HyperDE van a ser extendidos para dar soporte a las aplicaciones Web 2.0 (interfaces ricas, mashups, etc.).

3.2.4 UWE: UML based-web Engineering

UWE (UML base-web Engineering) [56, 68, 69] es un método para el desarrollo de aplicaciones Web basado totalmente en técnicas UML. Este método propone un proceso de diseño semi-automático soportado por la herramienta `ArgoUWE`[6].

[6] http://www.pst.informatik.uni-muenchen.de/projekte/uwe

Proceso de desarrollo

El proceso de desarrollo de UWE se basa en MDA y consta de 4 etapas:

- El proceso comienza en la etapa de *análisis de requisitos*. Estos requisitos se definen mediante diagramas de casos de uso.
- La etapa de *diseño conceptual* define los modelos conceptuales representados con diagramas de clase.
- El *diseño navegacional* está formado por las clases navegacionales y la estructura de navegación. Ambos diagramas se obtienen de los modelos de las etapas anteriores.
- El proceso termina con la etapa de *diseño de la presentación* donde se construye un modelo de presentación basado en los modelos anteriores.

Adaptación a los servicios Web

La propuesta no presenta una estrategia de modelado o de implementación de servicios Web. En [70] proponen, como trabajo futuro, el análisis de una aproximación MDE a características de la Web 2.0 tales como servicios Web y aplicaciones ricas utilizando tecnología AJAX. Además, en [69] muestra, con ejemplos, cómo sería el código de los servicios Web.

3.2.5 WebML: Web Modeling Language

WebML (Web Modeling Language) [57, 71, 19, 72] es un lenguaje de alto nivel de modelado y especificación para aplicaciones Web. WebML permite a los diseñadores expresar las principales características de un sitio a un alto nivel de abstracción y sin comentar detalles de arquitectura. Provee gráficos, formalismos, especificaciones, y diseño de procesos apoyados por herramientas gráficas.

WebRatio [7] es la herramienta que da soporte a este método. Esta her-

[7] Http://www.webratio.com

ramienta está siendo aplicada en entornos industriales.

Proceso de desarrollo

El proceso de desarrollo de WebML consta de seis etapas:

1. La primera etapa es la de *análisis de requisitos*. En esta etapa se especifica la información sobre la aplicación.

2. La etapa de *modelado conceptual* consiste en definir los esquemas conceptuales que expresan la organización de la aplicación en un alto nivel de abstracción, totalmente independiente de detalles de implementación.

3. La *implementación* consiste en la transformación de los modelos WebML en una implementación específica.

4. En la etapa de *prueba y evaluación*, la aplicación Web es probada y validada, tanto su calidad interna como externa.

5. El *despliegue* de la aplicación Web en una arquitectura específica.

6. La etapa de *mantenimiento y evolución* consiste en el mantenimiento de la aplicación tras su despliegue.

Adaptación a los servicios Web

WebML ha sido extendido para dar soporte al desarrollo de aplicaciones basadas en servicios Web [30][8], definiendo nuevas primitivas para la representación de éstos en el modelo de hipertexto. La extensión se basa en la implementación de WSDL [7], y se representan mediante las unidades de servicios Web.

WebML soporta los cuatro tipos de operaciones existentes en WSDL: `One-way`, `Request-Respon`, `Notification` y `Solicit & Response` (ver figura 3.3[9]).

[8] Estos trabajos son posteriores a las publicaciones iniciales de esta tesis [73, 74]

[9] Figura extraída del artículo [72]

Asumen que estas operaciones no se utilizan dentro de los esquemas tradicionales de hipertextos que representan la aplicación Web, sino como vistas de servicios con la definición de los servicios publicados. Las dos primeras operaciones estarían dentro de lo que WebML llama *integración con servicios Web* y las dos últimas con la *publicación de servicios Web*.

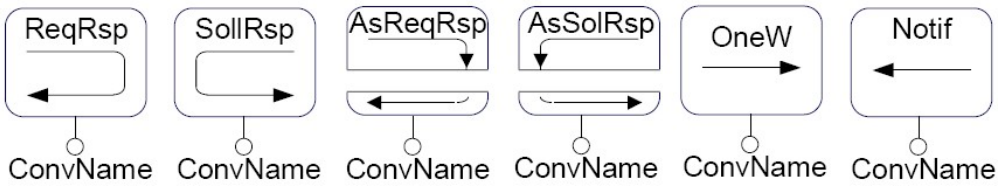

Figura 3.3: Modelado de servicios Web en WebML

Los dos últimos tipos de operaciones (`Notification` y `Solicit&Response`) permiten publicar los servicios Web para ser invocadas por otras aplicaciones. Por lo tanto se puede afirmar que WebML permite el diseño de servicios Web, pero no ofrece ninguna guía para la obtención de los mismos a partir de sus modelos.

Desde el punto de vista de implementación WebRatio si que permite generar servicios Web pero el despliegue y la publicación requiere la extensión de WebRatio [72].

Este trabajo difiere de la propuesta de la tesis en que WebML [75, 30] propone una extensión que permite modelar servicios Web y que genera automáticamente su implementación, pero no propone ningún método que permita derivar el diseño de los servicios Web a partir de los modelos conceptuales WebML.

3.2.6 WSDM: Web Site Design Method

WSDM (Web Site Design Modeling) [58, 76] es un método de diseño de sitios Web. WSDM centra la generación del diseño en el usuario más que en los datos. Para esto primero define los tipos de usuarios que pueden acceder al sitio Web, y a partir de esta definición se establecen los parámetros de diseño. No existe ninguna herramienta que de soporte al método WSDM.

Proceso de desarrollo

El método está compuesto por cinco etapas:

1. La primera fase es la *misión de la aplicación* donde se especifican los objetivos de la aplicación Web.
2. En la fase de *modelado de los usuarios* los usuarios se clasifican y agrupan para estudiar los requisitos para cada grupo de usuarios.
3. En el *diseño conceptual* se modela la información requerida y se detallan las clases de los usuarios. En el diseño conceptual también se crea el diseño de la navegación.
4. En la fase de *diseño de la implementación* se especifican los requisitos y restricciones del diseño gráfico de la aplicación Web, según lo definido en el diseño conceptual.
5. Finalmente, en la fase de *implementación* se selecciona la tecnología de desarrollo y se implementa la aplicación Web.

Adaptación a los servicios Web

WSDM permite el uso de servicios Web externos [77, 76] pero no soporta el diseño ni la implementación de los servicios Web propios. Esta propuesta no presenta un método para derivar los servicios Web a partir de sus modelos conceptuales.

3.3 Análisis comparativo de los principales métodos de Ingeniería Web

Una vez analizadas las soluciones propuestas por cada método, en este apartado se presentan las desventajas y puntos débiles encontrado en cada propuesta con su respectiva solución. Para llevar acabo esta evaluación, se van a tener en cuenta los siguientes aspectos:

1. **Diseño de los servicios Web**, si permiten el diseño de servicios Web y han extendido sus modelos con primitivas para ello.

2. **Implementación de los servicios Web**, si las herramientas que les dan soporte generan código para los servicios Web.

3. **Guía para derivar los servicios Web de los modelos conceptuales**, si existe alguna publicación donde se cuente cómo obtener las operaciones de los servicios Web a partir de los modelos conceptuales de cada método.

   Si los puntos (1) o (2) se han contestado afirmativamente, además se analiza si cada una de los métodos ofrece servicios Web que den soporte a:

4. **Lógica de Aplicación**, son operaciones que implementan los requisitos funcionales (lógica) de la aplicación.

5. **Gestión de Usuarios**, son operaciones que implementan la autenticación, autorización y gestión de los usuarios.

6. **Recuperación de Información**, son operaciones que implementan la búsqueda y recuperación de información.

7. **Soporte a la Navegación**, son operaciones que implementan la navegación permitida a cada uno de los usuarios del sistema.

8. **Soporte a la Presentación**, son operaciones que conocen cómo se debe presentar la información.

Diseño de servicios Web

La mayoría de los método analizados permiten el diseño de servicios Web e incluso han creado nuevos modelos y/o primitivas. Hera, OOHDM y WebML son los métodos que han incluido el diseño de servicios Web dentro de sus etapas de modelado de aplicaciones Web.

Por otro lado WSDM no ha incluido los servicios Web propios dentro de sus primitivas, pero sí los servicios Web ajenos. En cambio, UWE y OO-H no

ha incluido todavía primitivas ni modelos para incluir ningún tipo de servicio Web.

Implementación de servicios Web

Los métodos Hera, OOHDM y WebML ofrecen soporte a la generación de servicios Web a partir de sus modelos. OO-H en cambio, no permite el diseño en sus modelos pero si la implementación de sus aplicaciones como servicios Web. Por otro lado, ni UWE ni WSDM permiten generar servicios Web.

Guía para derivar los servicios Web de los modelos conceptuales

Como regla general, los método de ingeniería Web analizados no proveen ninguna guía para derivar los servicios Web de los modelos conceptuales propuestos en sus métodos. Dejan en manos de los analistas/modeladores la forma de diseñar los servicios Web.

Servicios Web de Lógica de la Aplicación

Tanto OO-H, como OOHDM y WebML dan soporte a los servicios Web que implementan la lógica de la aplicación. En el caso de OO-H se genera toda la capa de lógica de la aplicación como un servicio Web. En el caso de OOHDM y WebML solo se implementan aquellas operaciones que han sido modeladas como un servicio Web.

Servicios Web para la Gestión de Usuarios

WebML solo tiene dos de las operaciones existentes para la gestión de usuarios: `login` que da acceso al usuario a la aplicación y `logout` que permite salir de la sesión.

Servicios Web de Recuperación de Información

Tanto Hera como WebML permiten modelar e implementar servicios Web que recuperen información de las clases existentes en el modelo.

Servicios Web Soporte a la Navegación

Al igual que la guía para derivar los servicios Web, ninguno de los métodos analizados provee servicios que den soporte a la navegación modelado en sus métodos.

Servicios Web de Presentación

Únicamente Hera da soporte a los servicios Web de presentación, pero con la salvedad de que estos servicios Web no permiten al usuario configurar cómo se desea ver la información ya que no son dinámicos.

3.4 Conclusiones

En este capítulo se ha presentado una revisión del estado de la investigación relacionada con el diseño e implementación de servicios Web. Se han revisado los principales métodos de la Ingeniería Web, poniendo énfasis en cómo se han adaptado a los servicios Web. En primer lugar se ha introducido cada propuesta presentando su método de desarrollo. A continuación, se ha presentado cómo se han adaptado estos métodos a los servicios Web tanto a nivel de diseño como de implementación. Además, se ha analizado si estos método proveen alguna guía para derivar (automática o manualmente) los servicios Web de sus modelos conceptuales. Una vez los métodos han sido presentados, se ha realizado un estudio comparativo de las propuestas donde se han destacado las principales aportaciones del método que se propone en esta tesis doctoral.

Desde el punto de vista de los servicios Web, alguno de los métodos permiten el diseño de servicios Web. La mayoría de ellos permite la implemen-

tación de los servicios Web mediante las herramientas que les dan soporte, y ninguno de ellos provee de una guía para derivar los servicios Web desde los modelos conceptuales propuestos.

Una crítica general a todas las propuestas es que ninguna de ellas propone una guía para derivar automáticamente los servicios Web a partir de los modelos conceptuales que proponen. Además, tampoco proponen servicios Web que den soporte a la navegación definida en sus métodos.

La tabla 3.1 resume los aspectos estudiados para cada uno de los métodos.

Tabla 3.1: Resumen de los métodos de Ingeniería Web

	Métodos de Ingeniería Web					
	Hera	OO-H	OOHDM	UWE	WebML	WSDM
Diseño	Si	No	Si	No	Si	No
Implementación	Si	Si	Si	No	Si	No
Guía	No	No	No	No	No	No
Lógica Aplicación	No	Si	Si	-	Si	-
Gestión Usuarios	No	No	No	-	Si *	-
Recuperación Información	Si	No	No	-	Si	-
Soporte Navegación	No	No	No	-	No	-
Soporte Presentación	Si **	No	No	-	No	-

* Solo las operaciones de `login` y `logout`.

** Solo con presentación no configurable por el usuario (estática).

Capítulo 4

Un método para el diseño e implementación de servicios Web

En este capítulo se presenta un método dirigido por modelos para el diseño e implementación de servicios Web a partir de los modelos conceptuales de OO-Method / OOWS.

El capítulo se estructura de la siguiente forma: para comprender mejor la propuesta, en primer lugar se introduce brevemente la notación utilizada para describir el método junto con una visión general y completa del método para la identificación e implementación de servicios Web. El método se compone de cuatro etapas: la *Especificación de Requisitos* en el apartado 4.2, el *Modelado Conceptual* en el apartado 4.3, la *Generación de Código* en el apartado 4.4 y en la parte final del capítulo, en el apartado 4.5, se esboza la propuesta de la tesis para el diseño e implementación de Servicios Web (estos aspectos se desarrollan con mayor nivel de detalle en los capítulos 5 y 6 respectivamente).

Para ejemplificar los diferentes subapartados, se aplica la propuesta al ejemplo del apéndice A. En este ejemplo se modela el caso de estudio de una aplicación de comercio electrónico como *Amazon*, donde las contribuciones de esta tesis se ponen en práctica. En este caso de estudio se desarrolla una aplicación de comercio electrónico, mediante servicios Web. Se ha escogido una

aplicación de comercio electrónico porque este tipo de aplicaciones se centran tanto en proveer una gran cantidad de información como en proveer funcionalidad compleja para por ejemplo comprar productos. Estas propiedades permiten mostrar las características principales del método para el diseño y la implementación de servicios Web. Se ha utilizado una aplicación de comercio electrónico similar a Amazon para facilitar la comprensión del método, y porque Amazon es una de las aplicaciones de comercio electrónico más utilizadas y conocidas.

4.1 Introducción

Para definir las etapas involucradas en el método de diseño e implementación de servicios Web, se han estudiado varias propuestas (OPEN Process Framework [78], SPEM [34], Terävä [79] y PIE [80]). Se ha elegido SPEM v2.0 porque proporciona la abstracción suficiente para describir el método que se propone en la tesis.

4.1.1 Conceptos básicos de SPEM

SPEM v2.0 (*Software Process Engineering Metamodel*) [34], formulada por la OMG (Object Management Group), permite definir procesos software por medio de un conjunto de *actividades*. Cada una de estas actividades se lleva a cabo por uno o más *roles*. Tras la realización de cada actividad se pueden obtener uno o más *productos*, y una actividad puede también necesitar algún producto para llevarse a cabo. Para definir el proceso de trabajo y comunicación entre los roles, actividades y productos, SPEM utiliza las *categorías* o *etapas*. A continuación se detallan los elementos de SPEM y en la figura 4.1 se presenta la notación propuesta por SPEM para representar estos elementos.

- *Rol* (*processRol*): define el comportamiento y responsabilidades de un individuo, o de un grupo de individuos trabajando juntos como un equipo. Una persona puede desempeñar diversos roles, así como un mismo rol puede ser representado por varias personas.

- *Actividad* (*activity*): se define como una unidad de trabajo que se le solicita a una persona que está desempeñando un rol. Las actividades tienen un objetivo concreto, normalmente expresado en términos de crear o actualizar algún producto.

- *Producto* (*workProduct*): se define como una pieza de información que es producida, modificada o usada durante el proceso de desarrollo de software. Los productos son los resultados del proceso, que se van generando y usando hasta obtener el producto final. Un producto puede ser: un documento, un modelo o una aplicación (código fuente o ejecutable).

- *Categoría* o *Etapa* (*category*): divide las actividades dentro de un proceso teniendo en cuenta un tema común. Definen el proceso de trabajo y comunicación entre los roles, actividades y productos.

- *Guía* (*Guidance*): proporciona información adicional relacionada con los elementos.

- *Herramienta* (*Tool*): especifica la participación de una herramienta en la realización de una actividad.

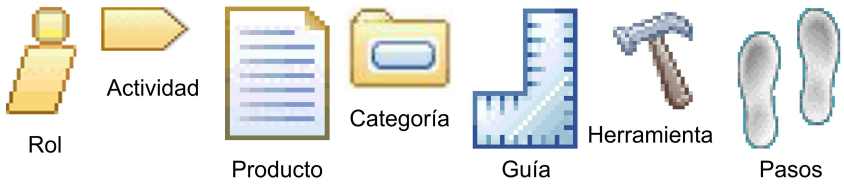

Figura 4.1: Representación gráfica de la notación en SPEM

Además, SPEM propone el uso de diagramas de actividad UML 2.0 [81] para definir tanto la secuencia de actividades como la salida y entrada de productos. En este caso, los nodos de los diagramas representan actividades o productos. Los arcos en los diagramas de actividades representan:

1. una *secuencia de actividades*, representado con flechas continuas, si tanto el origen como el destino son actividades,

2. *la salida o la entrada de productos*, representado con flechas discontinuas, si el origen o el destino de la flecha es un producto.

Figura 4.2: Ejemplos de conexión en SPEM

Finalmente, SPEM permite asociar elementos *guía* que se representan en los diagramas de actividad. Las actividades se asocian a elementos guía para indicar información detallada sobre cómo se llevan a cabo las actividades. En esta tesis, esta guía indica los diferentes capítulos de la tesis donde se introduce información relacionada con la actividad. En la figura 4.2 se puede ver un ejemplo de una actividad con guía.

4.1.2 Un Método para el diseño e implementación de servicios Web

En este apartado, se presenta el método OO-Method / OOWS extendido con la propuesta de esta tesis: el soporte para el diseño e implementación de servicios Web. Para ello, primero se muestra el estado actual de OO-Method / OOWS.

OO-Method [61] es un método Orientado a Objetos de producción automática de software basado en un modelo formal de objetos, desarrollado en el grupo de investigación al que pertenece la autora de la tesis. OOWS [12] es la extensión de OO-Method que introduce la expresividad necesaria para capturar los requisitos navegacionales y de presentación de las aplicaciones Web.

El proceso de desarrollo llevado a cabo en OO-Method / OOWS, se basa en el paradigma del Desarrollo Dirigido por Modelos (MDD, Model Driven Development) [10]. MDD propone desarrollar software a partir de modelos. Propone construir el modelo de un sistema para transformarlo en su producto

software equivalente.

A continuación se presentan las técnicas y los modelos de OO-Method / OOWS, que van a servir de base para el trabajo desarrollado en esta tesis. Trasladando OO-Method / OOWS a SPEM (las tres primeras etapas mostradas en la figura 4.3)), el método presentado en la tesis se define a partir de los modelos obtenidos (productos) en las siguientes tres etapas (como se muestra en la figura 4.3): *Especificación de Requisitos*, *Modelado Conceptual* y *Generación de Código*.

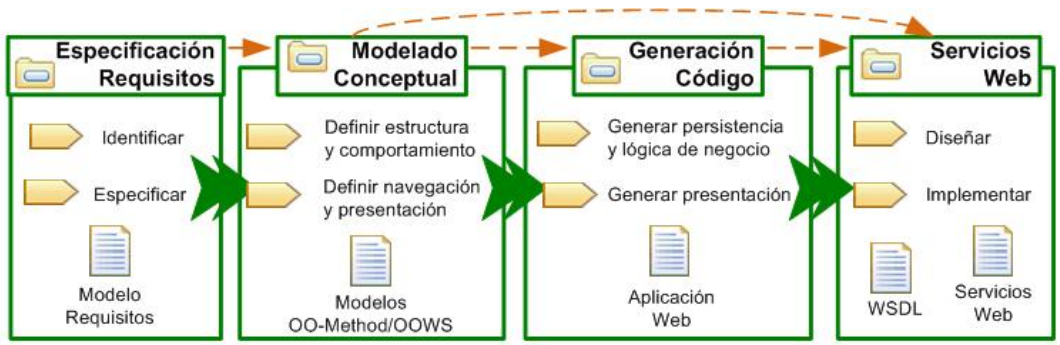

Figura 4.3: Visión general del método

La figura 4.4 muestra la secuencia de actividades para OO-Method / OOWS extendida con el diseño e implementación de los servicios Web.

La primera actividad comienza cuando el analista crea los modelos de requisitos tras analizar las necesidades del usuario. Los modelos creados por el analista son: el modelo de tareas, la plantilla de caracterización y la descripción de la tarea. En el apartado 4.2 se presenta esta actividad.

A continuación, el modelador define la estructura y el comportamiento de la aplicación a partir de los modelos de requisitos definidos por el analista en la actividad anterior. De esta actividad se obtienen los modelos conceptuales OO-Method de la aplicación (modelos estructural, dinámico y funcional). En un segundo paso, el modelador define la navegación y presentación abstracta de la aplicación Web obteniendo los modelos conceptuales OOWS (modelos de usuario, navegacional y de presentación). En el apartado 4.3 se desarrolla esta actividad.

Una vez modelada la aplicación, a partir de los modelos OO-Method, la herramienta OlivaNOVA genera automáticamente una aplicación software. A su vez, la herramienta OOWS Suite automáticamente genera, a partir de los modelos OOWS, la interfaz Web de dicha aplicación. En el apartado 4.4 se desarrolla la estrategia de generación automática de código.

Finalmente, se diseñan las operaciones de los servicios Web partiendo de los modelos de requisitos (modelos obtenidos en la primera etapa) y de los modelos conceptuales, de navegación y de presentación (modelos obtenidos en la segunda etapa). A continuación se implementan estas operaciones utilizando la lógica generada tanto por la herramienta OlivaNOVA como por la herramienta OOWS Suite. Estas dos actividades se han automatizado en la herramienta `DISWOOM`. Esta secuencia de actividades se ve detalladamente en capítulos posteriores (el diseño de los servicios Web en el capítulo 5, la implementación en el capítulo 6 y la herramienta en el capítulo 7).

Como se puede observar en la figura 4.4, existen dependencias entre las distintas actividades:

- La etapa de *Modelado Conceptual* necesita los modelos de la etapa de Especificación de Requisitos para modelar la estructura y el comportamiento de la aplicación.
- La etapa de *Generación de Código* necesita los modelos de la etapa de Modelado Conceptual. Por un lado, la herramienta OlivaNOVA depende de los modelos OO-Method. Por otro lado, la herramienta OOWS Suite depende de los modelos OOWS.
- La etapa de *Diseño e Implementación de servicios Web* necesita para diseñar las operaciones de los servicios Web: (1) los modelos de la etapa de Especificación de Requisitos; y (2) los modelos de la etapa de Modelado Conceptual: clases, usuario, navegación y presentación. Y para implementar las operaciones de los servicios Web: (1) la lógica y persistencia generada por la herramienta OlivaNOVA; y (2) la lógica de la capa de presentación generada por la herramienta OOWS Suite.

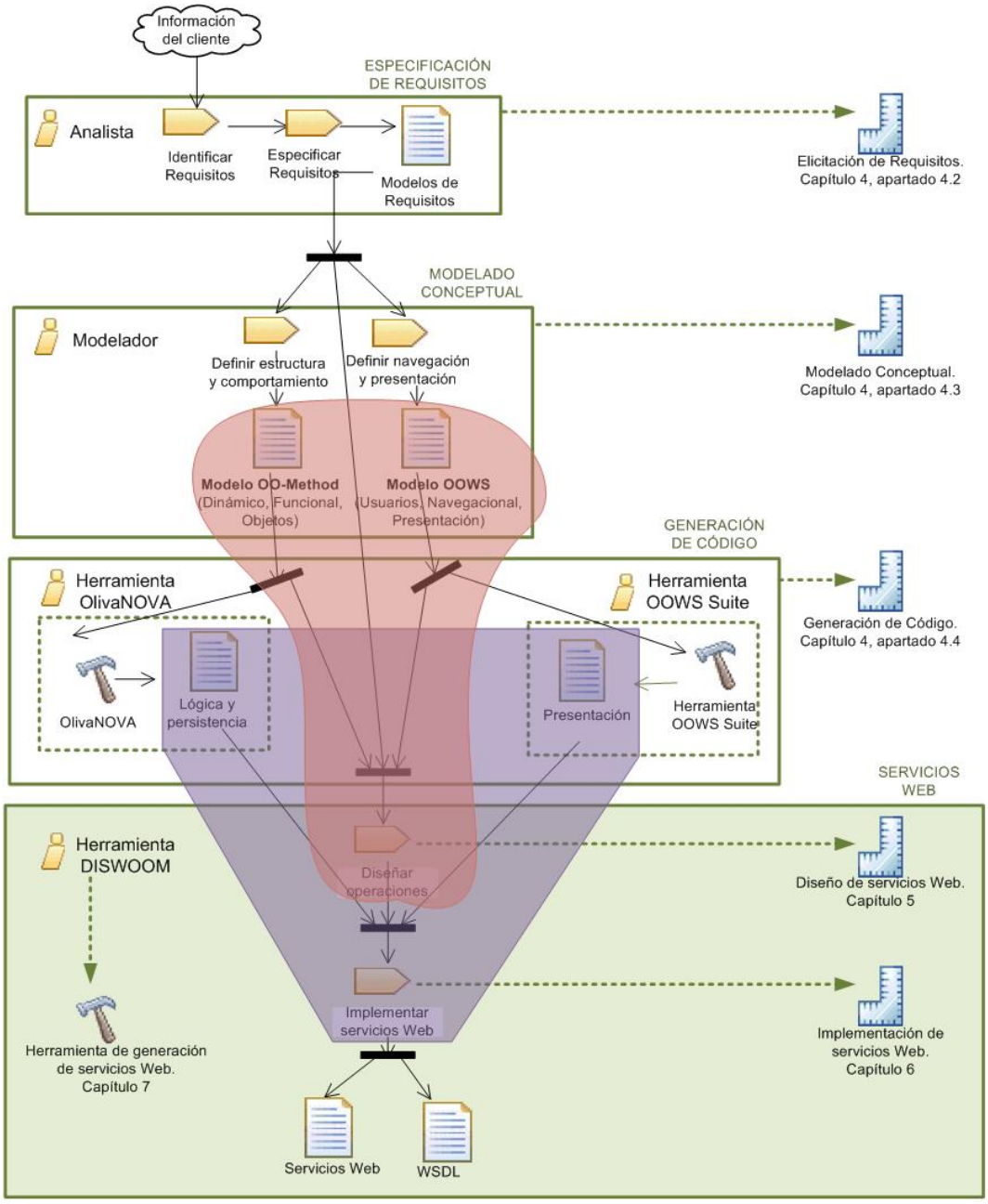

Figura 4.4: Secuencia de Actividades

4.2 Especificación de Requisitos

A continuación se presenta la etapa de Especificación de Requisitos [82] que constituye una etapa básica en el método que se propone.

La Especificación de Requisitos constituye la etapa temprana en el método de desarrollo OO-Method / OOWS. Esta etapa tiene como objetivo analizar y documentar las necesidades funcionales que deberán ser soportadas por el sistema a desarrollar. En esta etapa, se capturan los requisitos del sistema a partir de la descripción de las tareas que los usuarios deben poder realizar usando la aplicación Web. Para ello, se propone un modelo de requisitos basado en el concepto de tarea [82]. En la figura 4.5 se pueden observar las actividades, roles y productos que se desarrollan en esta etapa.

Existen más propuestas, a parte de OO-Method / OOWS, que utilizan los modelos de tareas. De entre los métodos de Ingeniería Web se encuentra WHDM [83], mientras que en la comunidad HCI existen diversos trabajos sobre los modelos de tareas [84, 85, 86].

Figura 4.5: Actividades, Roles y Productos de la *Especificación de Requisitos*

4.2.1 Actividades

Las actividades a llevar a cabo por el rol *Analista* son la identificación de los requisitos que ha de satisfacer el nuevo sistema mediante entrevistas, el estudio de los problemas del cliente y sus necesidades actuales. Estas actividades se realizan siguiendo el proceso presentado en [82]. Las actividades de esta etapa se resumen en dos:

1. *Identificar los requisitos del usuario*, a partir de la información proporcionada por el cliente se extraen las necesidades que debe cubrir la aplicación.
2. *Especificar y describir los requisitos*, a partir de las necesidades del cliente se elabora un catálogo (modelo) de requisitos.

Identificar los requisitos del usuario

Esta actividad tiene como finalidad capturar los requisitos de los usuarios para el desarrollo del sistema. La entrevista consiste en una interacción con un usuario para extraer los conocimientos de éste. Esta actividad comprende:

- Planificar las entrevistas a realizar.
- Realizar las entrevistas y documentarlas debidamente.
- Documentar los requisitos identificados con sus prioridades.

A partir de las entrevistas realizadas con los responsables y usuarios, se identifican los requisitos que debe cumplir el sistema y se establece una prioridad para los mismos, de acuerdo a las necesidades expresadas por los usuarios y a los objetivos a cubrir por el nuevo sistema.

Especificar y describir los requisitos

Esta actividad tiene como objetivo la especificación de forma clara, precisa y completa todas las funcionalidades y restricciones del sistema que se desea

construir. De las necesidades detectadas y de las peticiones de los usuarios se elabora un Catálogo de Requisitos formado por los tres productos de esta actividad: modelo de tareas y las descripciones de cada tarea (ver figura 4.6).

Esta documentación está sujeta a revisiones por el grupo de usuarios que se recogen por medio de sucesivas versiones del documento, hasta alcanzar su aprobación por parte del grupo de usuarios. Una vez aprobado, sirve de base para la construcción del nuevo sistema.

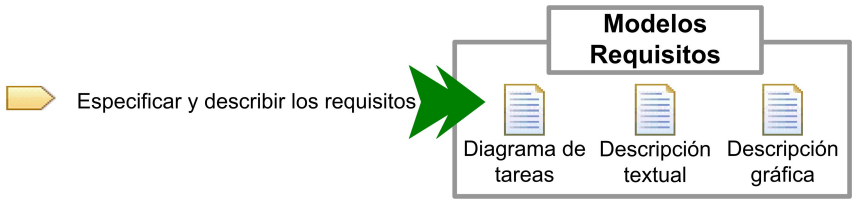

Figura 4.6: Modelos obtenidos en la especificación y descripción de requisitos

En este apartado se han presentado las actividades que debe realizar un Analista para especificar y describir los requisitos de la aplicación. A continuación, se presentan los productos que se obtienen al llevar a cabo estas actividades.

4.2.2 Productos

En este apartado se introducen los productos de la etapa de especificación de requisitos que se van a utilizar en el desarrollo de la tesis. El producto que se obtiene de estas actividades es un modelo de requisitos formado por: un modelo de tareas para cada usuario y una descripción para cada tarea elemental del modelo de tareas.

Modelo de tareas

El primer producto consiste en construir, a partir del propósito general del sistema, la taxonomía de tareas que caracteriza la aplicación en desarrollo. La taxonomía de tareas se construye agrupando las tareas generales, a partir de las

cuales, es posible llevar a cabo un refinamiento obteniendo tareas más específicas, hasta obtener tareas elementales. En este sentido, una tarea elemental se define como aquella que al particionarla en subtareas, se obtienen tareas cuyo trabajo implica únicamente la participación del sistema o del usuario, no de ambos.

En definitiva, se define un *modelo de tareas* para cada tipo de usuario (ver figura 4.7). En este modelo se describe de forma jerárquica las tareas que cada usuario puede llevar a cabo interactuando con la aplicación. Para representar el modelo de tareas se utiliza el modelo de tareas propuesto en [87] (basado en la aproximación CTT (ConcurTaskTree) [88]).

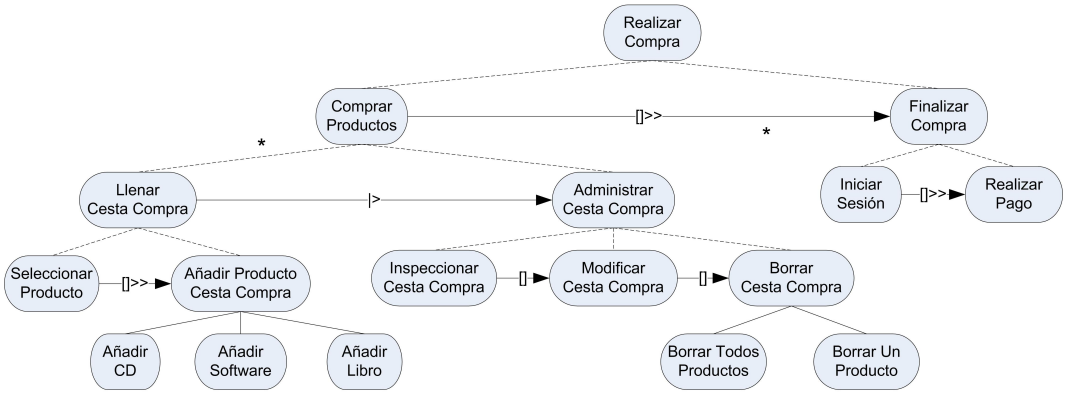

Figura 4.7: Modelo de tareas

Descripción de cada tarea elemental

A cada tarea elemental del modelo de tareas se le asocian dos descripciones (información más detallada se puede encontrar en [89, 90]):

- **Plantilla de Caracterización**. La plantilla de caracterización asociada a cada tarea elemental, muestra información de por qué se realiza la tarea, quien la lleva a cabo y con qué frecuencia. Por lo tanto, para cada tarea elemental hay que especificar la siguiente información (ver figura 4.8):
  - *Task*: nombre de la tarea.

- *Goals*: los objetivos que se deben alcanzar al realizarla.
- *Users*: los usuarios que pueden llevarla a cabo.
- *Frequency*: frecuencia de uso.

Task:	Añadir Cesta
Goals:	El usuario añade un producto a su carrito de la compra
Users:	Anónimo, Cliente
Frequency:	100 veces por hora

Figura 4.8: Plantilla de caracterización de una tarea elemental

- **Descripción de la tarea**. A cada tarea elemental del modelo de tareas se le asocia también una descripción (ver figura 4.9). Este tipo de descripciones introducen información sobre la propia interacción entre el usuario y el sistema, indicando de forma explicita en qué momento de la tarea se lleva a cabo esta interacción. En este contexto, una tarea se describe como un proceso donde el sistema lleva a cabo determinadas acciones deteniéndose en algunos momentos para interactuar con el usuario a través de *puntos de interacción (PI)*.

  Para realizar la descripción de tareas elementales mediante PIs, se utilizan diagramas de actividad basados en los diagramas de actividad de UML [81] donde cada nodo representa un PI (línea continua) o una acción del sistema (línea discontinua).

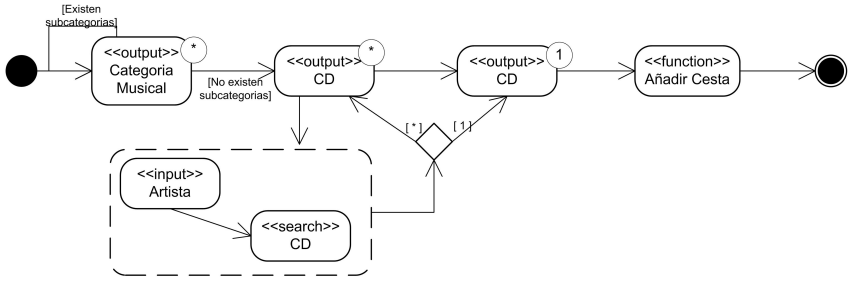

Figura 4.9: Descripción de una tarea elemental

La interacción entre el sistema y el usuario se representa explícitamente mediante PIs donde el sistema proporciona o solicita información al usuario. Dependiendo de esto, los PI se clasifican en:

- *PI de salida* (estereotipados con la palabra clave «*output*»): el sistema proporciona al usuario información y acceso a unas determinadas operaciones. En este tipo de puntos, se indica en la parte superior derecha, el número de instancias sobre las que se informa (cardinalidad).
- *PI de entrada* (estereotipados con la palabra clave «*input*»): el usuario introduce información en el sistema sobre una determinada instancia. Este PI depende exclusivamente de la acción del sistema y no interviene directamente en el proceso general que permite alcanzar los resultados asociados a la tarea. Para caracterizar esto, los nodos que representan a ambos elementos se encapsulan dentro de un cuadro punteado, como se observa en la figura 4.9.

Además, se pueden indicar las acciones que lleva a cabo el sistema, clasificadas en:

- *Ejecución de funcionalidad* (estereotipadas con la palabra clave «*function*»): operaciones del sistema que cambian su estado (alta, baja, modificación, etc.).
- *Búsqueda de información* (estereotipadas con la palabra «*search*»): operaciones que únicamente consultan el estado del sistema.

4.3 Modelado Conceptual

En este apartado se presentan las actividades y productos de la etapa de Modelado Conceptual. En esta etapa, se describe a un alto nivel de abstracción, el sistema que da soporte a las tareas de usuario descritas en la etapa anterior. Estas abstracciones son especificadas en términos de clases y de su estructura, comportamiento y funcionalidad.

4.3.1 Actividades

Esta etapa incluye aquellas actividades relacionadas con el modelado de aplicaciones Web. Estas actividades se realizan siguiendo el proceso de desarrollo propuesto en OO-Method / OOWS. OO-Method [61] es un método de ingeniería que provee soporte metodológico para el desarrollo de aplicaciones tradicionales (no-Web), mientras que OOWS [12] es la extensión de OO-Method para el desarrollo de aplicaciones Web.

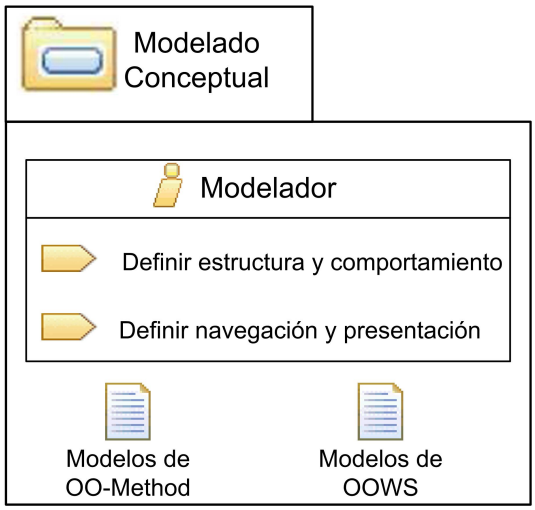

Figura 4.10: Actividades, Roles y Productos del *Modelado Conceptual*

En esta etapa se distinguen las siguientes actividades principales (ver figura 4.10):

1. *Definir la estructura y el comportamiento de la aplicación* (basada en OO-Method), que representa los requisitos de la aplicación (desde el modelo de requisitos definido por el analista en la etapa anterior).
2. *Definir la navegación y presentación de la interfaz Web* (basada en OOWS), a partir de los modelos de requisitos (etapa anterior) y del modelo de clases (modelo de objetos de la actividad anterior).

El rol que realiza estas actividades es el de Modelador. Los productos que se obtienen en esta etapa son un conjunto de modelos (objetos, dinámico,

funcional, usuarios, navegacional y de presentación) que capturan la estructura, el comportamiento, la navegación y la presentación de una aplicación a nivel conceptual.

Definir la estructura y el comportamiento de la aplicación

La primera actividad se basa en el método de OO-Method [61]. Este método propone la construcción de modelos para desarrollar un sistema software. Estos modelos se corresponden con el producto obtenido en la primera actividad (ver figura 4.11):

- El *modelo de objetos* describe la estructura estática de la aplicación (clases, operaciones y atributos) y las relaciones entre las clases (especialización, asociación y agregación) por medio de un modelo de clases.
- El *modelo dinámico* describe las vidas válidas para los objetos de una clase del sistema usando diagramas de transición de estados, ordenando temporalmente la secuencia de ocurrencia de servicios (eventos y transacciones). Además, en este modelo se especifica la interacción entre objetos (comunicación entre objetos) representada por medio del diagrama de secuencia.
- El *modelo funcional* permite especificar la semántica de los cambios de estado, definiendo cual es el efecto sobre el estado del objeto implicado en términos de los atributos que modifica.

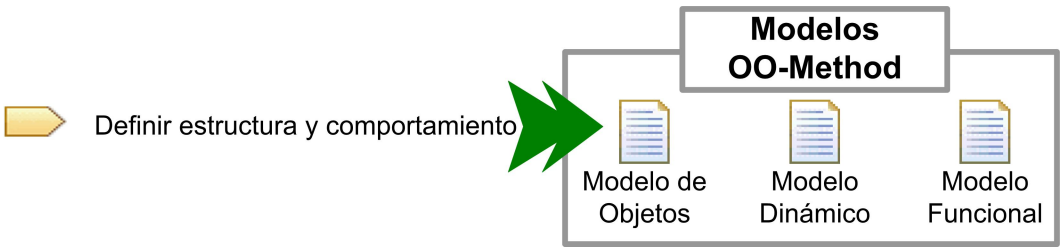

Figura 4.11: Modelos obtenidos en la primera actividad

La perspectiva estática del modelo de objetos se complementa con los modelos dinámico y funcional. Estos tres modelos definen, desde puntos de vista complementarios, las propiedades estáticas y dinámicas de los objetos identificados en el sistema.

Definir la navegación y presentación de la interfaz Web

La segunda actividad se basa en OOWS [12], el cual introduce tres modelos para capturar los nuevos aspectos que intervienen en el desarrollo de aplicaciones Web (ver figura 4.12):

- El *modelo de usuarios* describe los diferentes tipos de usuarios (roles) que van a interactuar con la aplicación Web.
- El *modelo navegacional* describe la estructura de la navegación permitida para los usuarios definidos anteriormente.
- El *modelo de presentación* define las propiedades de presentación de información de la aplicación Web.

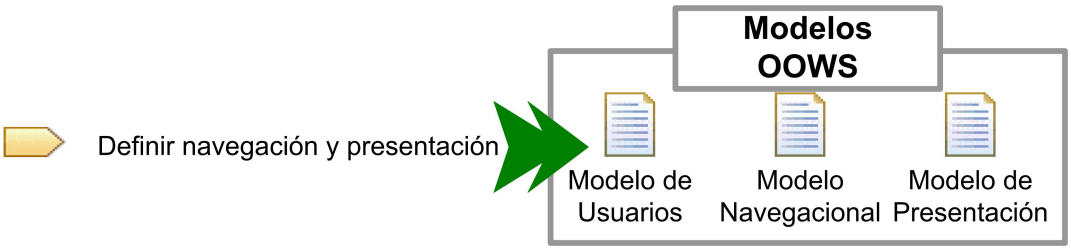

Figura 4.12: Modelos obtenidos en la segunda actividad

El método para el desarrollo de servicios Web a partir de modelos conceptuales que se presenta en esta tesis, no hace uso de todos los modelos propuestos en esta etapa. Por lo tanto, a continuación se detallan los modelos que proporcionan información relevante para la identificación e implementación de los servicios Web.

4.3.2 Productos

En este apartado se presentan el modelo de objetos (de la primera actividad de la etapa de Modelado Conceptual) y los modelos de usuario, navegación y presentación (de la segunda actividad).

Modelo de Objetos

El modelo de objetos captura la estructura estática del sistema mostrando: clases, relaciones entre clases, atributos y operaciones (ver figura 4.13).

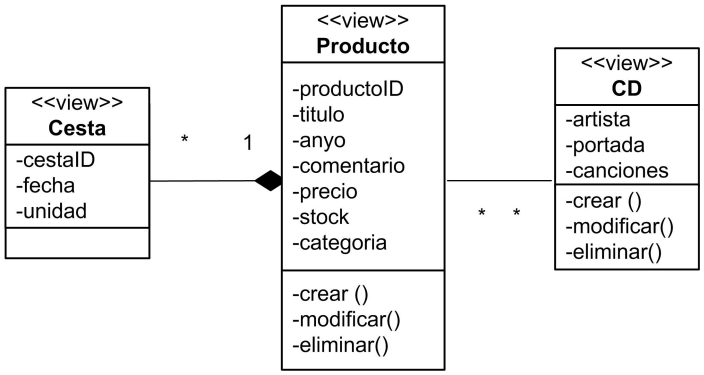

Figura 4.13: Modelo de clases

A continuación, se muestran los modelos de la segunda actividad que nos proporcionan información relevante para la identificación e implementación de los servicios Web. Los modelos OOWS permiten captar los requisitos de navegación de la aplicación.

Modelo de Usuarios

En el modelo de usuarios se especifica qué tipos de usuarios (roles de usuarios) van a interactuar con el sistema, qué interrelaciones existen entre ellos y cuál va a ser su modo de acceso al sistema.

En primer lugar se estudian cuáles son los tipos de roles de usuarios que van a interactuar con el sistema y se crea un rol de usuario en este modelo (ver figura 4.14). Cada tipo de rol de usuario puede ser de dos clases distintas:

- *Anónimo* (representados con el símbolo *?*): pueden conectarse al sistema sin identificarse. En el ejemplo de la figura 4.14 se ha definido un rol de usuario anónimo: *Visitante*.
- *Registrado* (representados con un candado): necesitan identificarse en el sistema. En el ejemplo de la figura 4.14 se han definido dos roles de usuario anónimo: *Cliente* y *Administrador*.

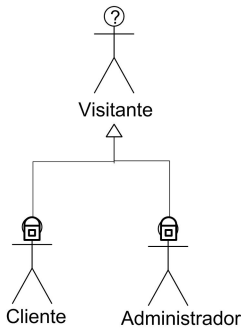

Figura 4.14: Modelo de Usuarios

Modelo Navegacional

En este modelo se construyen los mapas navegacionales, uno por cada rol de usuario. Estos mapas se definen mediante un grafo dirigido (ver figura 4.15) en el que los nodos representan unidades de interacción con el usuario (gráficamente representado mediante un paquete UML) y los arcos representan los caminos navegacionales válidos entre nodos (representados mediante flechas).

Cada uno de estos nodos, donde el usuario interactúa con el sistema, se corresponde con un contexto navegacional (ver figura 4.16). Los enlaces navegacionales (arcos del grafo) representan la alcanzabilidad entre nodos, indicando los caminos navegacionales válidos por el sistema para un determinado usuario. Los enlaces pueden ser de dos tipos:

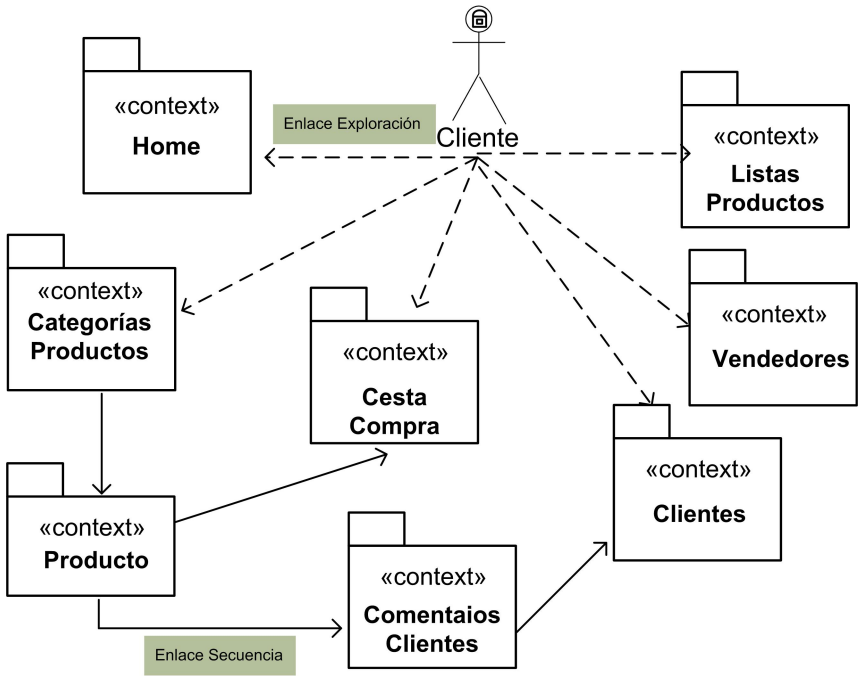

Figura 4.15: Mapa Navegacional

- *Enlaces de exploración* (se representa gráficamente mediante flechas discontinuas): indican contextos alcanzables desde cualquier otro contexto. Este tipo de enlaces surgen del rol del usuario hasta el contexto.

- *Enlaces de secuencia* (gráficamente se representa mediante flechas continuas): indican aquellos contextos que sólo pueden alcanzarse a través de un determinado camino de navegación.

Una vez se ha definido la navegación de cada uno de los roles identificados, mediante sus respectivos mapas navegacionales, el siguiente paso a realizar consiste en la definición de los contextos navegacionales.

Un contexto navegacional (ver figura 4.16), estereotipado con la palabra reservada << *context* >>, está compuesto por un conjunto de clases navegacionales, estereotipadas con la palabra reservada << *view* >> que hacen referencia a clases identificadas en el modelo de clases. Con ellas se puede definir la visibilidad (vista) ofertada al rol en este nodo, tanto de los atrib-

utos de la clase como de las operaciones que puede activar. En un contexto navegacional debe aparecer al menos una clase navegacional principal, llamada clase directora, y opcionalmente otras que complementan la información de esta clase, llamadas clases complementarias. Las clases navegacionales están unidas entre sí por relaciones binarias unidireccionales que pueden ser definidas sobre una relación de agregación o de especialización/generalización existente entre las dos clases en el modelo de clases.

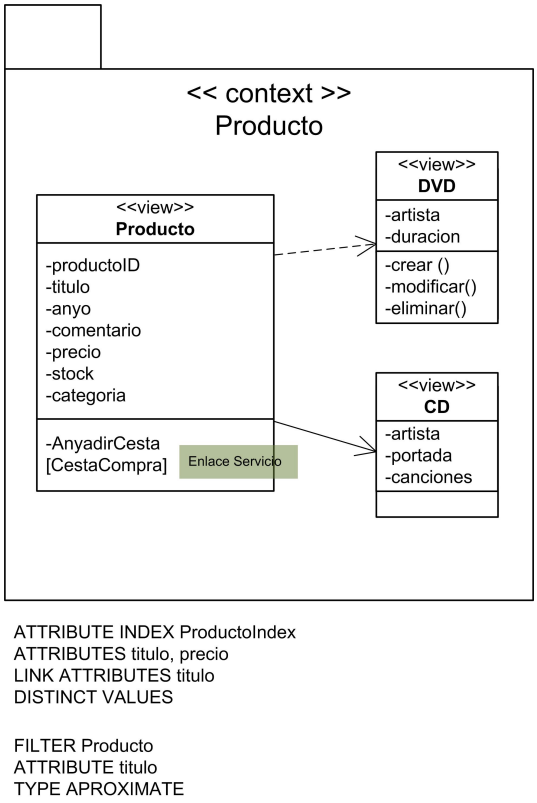

Figura 4.16: Contexto Navegacional

Se pueden definir otro tipo más de enlace, pero esta vez a nivel de contexto:

- *Enlaces de servicio*: permiten definir el contexto navegacional al cual accede el usuario después de la ejecución de una operación. Para definir este tipo de enlace es necesario indicar el contexto destino de la navegación (representado como *[contexto_destino]*).

Una vez construidos los mapas navegacionales se pueden definir mecanismos adicionales que estructuren el acceso y permitan realizar búsquedas de información dentro de un contexto navegacional. Estos mecanismos permitirán explorar y facilitar el acceso a la misma información, sin necesidad de implicar navegación. Estos mecanismos son los siguientes:

- *Índices*: proporcionan un acceso indexado (por alguna propiedad propia o de un objeto relacionado) a los objetos principales del contexto (objetos de la clase directora). Los índices crean una lista de información resumida permitiendo al usuario elegir una instancia de la lista.
- *Mecanismos de búsqueda*, expresados como filtros de información. Esta expresión se puede especificar en tiempo de Modelado o puede ser introducida por el usuario en tiempo de ejecución. Se pueden definir tres tipos de búsqueda en los filtros:
  - *Exacto*: devuelve el conjunto de instancias de la clase directora cuyo valor de atributo coincida exactamente con el valor indicado.
  - *Aproximado*: devuelve el conjunto de instancias de la clase directora cuyo valor de atributo sea semejante al valor indicado.
  - *Rango*: devuelve el conjunto de instancias de la clase directora cuyo valor de atributo esté entre un par de valores máximo y mínimo.

Modelo de Presentación

Una vez definido el modelo de navegación que captura la semántica navegacional del sistema, se deben asociar características de presentación al sistema. El modelo de presentación permite devolver los datos con una presentación definida por el usuario. Este modelo, complementa la información del modelo navegacional para la creación de interfaces con propiedades de presentación, donde se utilizan los contextos navegacionales como entidades básicas (ver figura 4.17).

Los requisitos de presentación se especifican por medio de patrones de presentación que se asocian a las primitivas de los contextos navegacionales (clases y relaciones). Los patrones de presentación que se pueden especificar son:

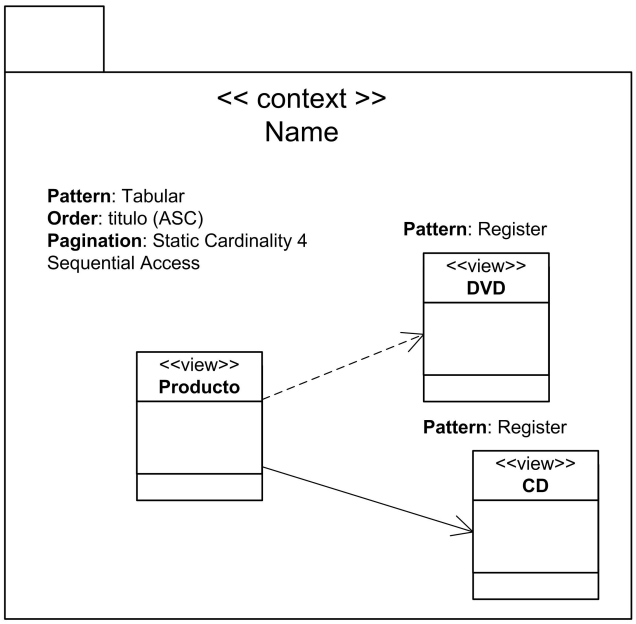

Figura 4.17: Contexto de presentación

- *Paginación de información (Pagination)*: permite capturar la semántica de desplazamiento de la información (*scrolling*). Permite definir cómo se quiere partir la información en bloques lógicos, de tal forma que en una pantalla solo aparezcan unas cuantas instancias del conjunto de todas las posibles. Además se proporcionan mecanismos para avanzar y retroceder entre las distintas páginas lógicas que se obtienen. Se define mediante:
  - la *cardinalidad*, que representa el número de instancias a mostrar en cada bloque; y
  - el *modo de acceso*, que define cómo se accede a los bloques (secuencial o aleatorio).

- *Ordenación (Order)*. Este patrón permite definir una ordenación de la población de una clase según el valor de uno o más atributos sobre los que se aplica. La información se muestra ordenada ascendentemente (ASC) o descendentemente (DES). Este patrón se puede aplicar tanto a clases navegacionales como a los índices y filtros de búsqueda mencionados anteriormente.

- *Presentación de instancias (Pattern).* Este patrón permite definir cómo se muestran los bloques de información. Existen 4 modos: registro, tabular, maestro-detalle y árbol.

4.4 Generación de Código

En este apartado se presenta la etapa de Generación de Código. Esta etapa sigue una estrategia de generación automática de código, donde se obtiene un producto software equivalente a la especificación realizada en la etapa de Modelado Conceptual.

4.4.1 Actividades

Esta etapa incluye las actividades relacionadas para la generación automática de código a partir de modelos conceptuales (productos de la etapa anterior). Se distinguen dos actividades principales (ver figura 4.18):

1. *Generación de la capa de persistencia y lógica de negocio* a partir de los modelos OO-Method obtenidos en la primera actividad de la etapa anterior.

2. *Generación de la capa de presentación* a partir de los modelos OOWS obtenidos en la segunda actividad de la etapa anterior.

Los roles que llevan a cabo estas actividades son las herramientas que dan soporte a los métodos OO-Method (OlivaNOVA [91, 92]) y OOWS (OOWS Suite [93, 12]). El producto que se obtiene después de llevar a cabo estas actividades, es un producto software que se implementa siguiendo una arquitectura de tres capas (ver apartado 4.4.2). Este producto software es equivalente a la especificación realizada en las dos etapas anteriores.

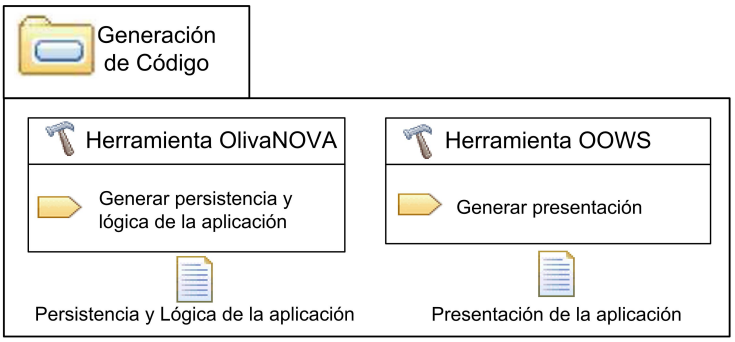

Figura 4.18: Actividades, Roles y Productos de la *Generación de Código*

Generación de la capa de persistencia y lógica de negocio

Por medio de la primera actividad se genera la información y la funcionalidad (las capas de Persistencia y Aplicación respectivamente) del producto software a partir de los modelos conceptuales de OO-Method (modelos de objetos, dinámico y funcional).

La herramienta OlivaNOVA Model Execution [91, 92] es la encargada de generar estas dos capas de la aplicación. OlivaNOVA genera la funcionalidad de la aplicación mediante componentes EJB y .NET, además de los esquemas de la base de datos utilizada por el sistema.

Generación de la capa de presentación

Por medio de la segunda actividad se genera la estructura navegacional (la capa de presentación) a partir de los modelos conceptuales de OOWS (modelos de usuario, navegacional y de presentación).

La herramienta OOWS Suite [93, 12] es la encargada de generar esta capa y permite acceder a la funcionalidad generada por OlivaNOVA (a las capas de Aplicación y Persistencia) a partir de un conjunto de páginas Web. Para acceder a la capa de Aplicación de una manera sencilla, OOWS Suite ha implementado un framework que ha denominado WIF (Web Interface Framework [94], se ve en detalle en el apartado 6.2, página 144) que permite ser reutilizado

por aplicaciones externas (como es el caso de los servicios Web presentados en esta tesis).

4.4.2 Productos

El producto software obtenido en esta etapa es implementado siguiendo una arquitectura de tres capas generadas por OlivaNOVA y OOWS Suite (ver figura 4.19):

- El *nivel de presentación*, donde se incluyen los componentes de interfaz de usuario (páginas Web y elementos/controles visuales) para interactuar y mostrar el resultado de las peticiones al nivel de aplicación.
- El *nivel de aplicación*, implementa la lógica de negocio del sistema mediante la implementación de las clases presentes en el Modelado Conceptual.
- El *nivel de persistencia*, el cual implementa el acceso a datos y la persistencia de los mismos, ocultando los detalles de los repositorios de datos a los niveles superiores.

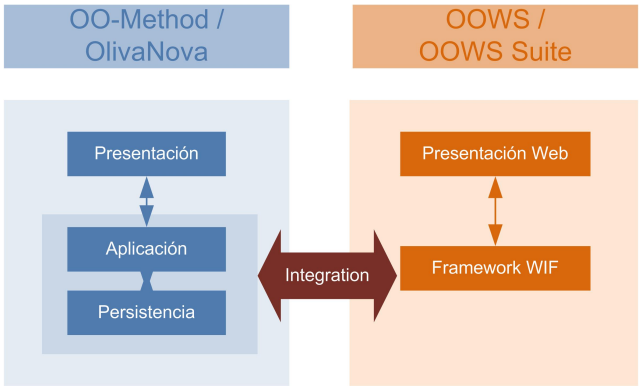

Figura 4.19: Capas de la aplicación generada

A continuación, se van a presentar en detalle los productos generados por cada una de las herramientas.

Productos generados por OlivaNOVA

La lógica de negocio modelada en OO-Method se transforma en OlivaNOVA a través de un motor de transformación concreto. Estos motores denominados OlivaNOVA Transformation Engines [92], son implementaciones de diferentes procesos de compilación de modelos conceptuales OO-Method. Cada motor, implementa:

- un repositorio del modelo conceptual,
- un repositorio del modelo de aplicación,
- un conjunto de correspondencias entre los elementos del repositorio del modelo conceptual y los elementos del repositorio del modelo de aplicación,
- y el conjunto de transformaciones asociadas a cada uno de los elementos del repositorio del modelo de aplicación.

El repositorio del modelo conceptual es común a todos los *transformation engines* y es capaz de cargar un modelo conceptual a partir de un fichero XML de intercambio producido por OlivaNOVA Modeler [91]. Las correspondencias acceden al repositorio del modelo conceptual y se encargan de ir creando elementos en el repositorio del modelo de aplicación. El repositorio del modelo de aplicación contiene elementos para cada uno de los elementos del modelo conceptual. También contiene los elementos necesarios para implementar la estrategia de ejecución de aplicaciones en la plataforma software específica del *transformation engine*, y los requeridos por la propia plataforma software. Las transformaciones acceden al repositorio del modelo de aplicación y generan la porción de código de la aplicación correspondiente a cada elemento del repositorio. En términos MDA, OlivaNOVA Transformation Engines es una implementación de una herramienta que realiza las transformaciones de PIM a PSM y las transformaciones de PSM a PIM (*Implementation Model*).

El resultado de un *transformation engine* varía dependiendo de la tecnología destino seleccionada (.NET, J2EE). Sin embargo la arquitectura de la aplicación se divide en dos componentes claramente diferenciados, un componente cliente, que encapsula la interfaz modelada mediante el Modelo de

Presentación OO-Method y un componente servidor, que encapsula toda la funcionalidad modelada (modelo dinámico y funcional) junto al acceso a la Base de datos (modelo estructural). Por lo tanto una aplicación OlivaNOVA está formada por dos componentes basados en el mismo modelo conceptual, pero independientes desde el punto de vista de la implementación. De esta característica se sacará provecho en el capítulo de generación de código (ver el capítulo 6 en la página 141).

Productos generados por OOWS Suite

Una vez definidos los modelos conceptuales que especifican una aplicación Web (modelo de usuarios, modelo de navegación y modelo de presentación), el siguiente paso es compilar estos modelos para producir el código equivalente. Esta acción se realiza con OOWS Suite, la herramienta que da soporte al método OOWS.

El *Framework WIF* (Web Interface Framework) es un framework que abstrae los conceptos tecnológicos relacionados con la aplicación Web a construir. Este framework se encarga de definir una interfaz Web que se comunica con la lógica de aplicación generada por OlivaNOVA (ver figura 4.20) y presenta una visión más «declarativa» de la construcción de interfaces Web. En vez de sustentarse en un lenguaje de programación, el framework se define sobre una serie de primitivas de alto nivel que representan conceptos habituales de las interfaces Web.

OOWS como método Web, permite definir el esquema conceptual de una aplicación Web. Sin embargo, tomando únicamente los modelos de OOWS, el resultado es una interfaz Web. Por otro lado, son los modelos conceptuales definidos mediante OO-Method quienes definen la funcionalidad del sistema. La comunicación con dicha lógica de aplicación, se produce mediante una fachada de negocio que se encarga de presentar la lógica cómo un conjunto de servicios de recuperación de información y de funcionalidad. El framework asume que la información se encuentra representada mediante un modelo del dominio, como un modelo de clases o de entidad/relación, de forma implícita o abstraída mediante la fachada de negocio. Como consecuencia, los atributos de las clases/entidades definen la información almacenada, y las operaciones la funcionalidad ofrecida. En el caso particular de definir la lógica con OlivaNO-

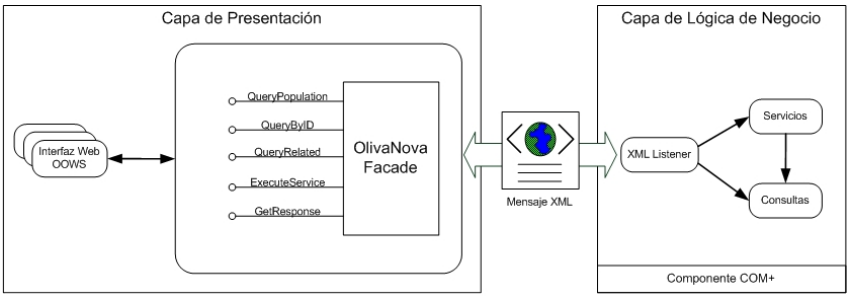

Figura 4.20: Integración con la lógica de negocio de OlivaNOVA

VA, la implementación comercial de OO-Method, esta fachada de negocio se comunica con un objeto COM+ que representa la información mediante un modelo de estructura (modelo de clases).

Una aplicación Web creada mediante el framework de interfaces Web se fundamenta en cuatro clases principales:

- `Application`, almacena la estructura navegacional de la aplicación Web y sus características globales.
- `Page`, define una página Web a la cual accederá el navegador.
- `Zone`, especifica el contenido que forma parte de una página.
- `Service`, establece el vínculo con la ejecución de la funcionalidad de la aplicación Web.

A partir de estas cuatro clases principales se definen una serie de métodos que permiten construir una interfaz Web que se presentan en detalle en el capítulo 6.

4.5 Diseño e Implementación de servicios Web

En este apartado se comentan brevemente las actividades y productos de la etapa de Diseño e Implementación de servicios Web, presentadas en la tesis.

En esta etapa se sigue una estrategia de generación automática de código a partir de modelos conceptuales.

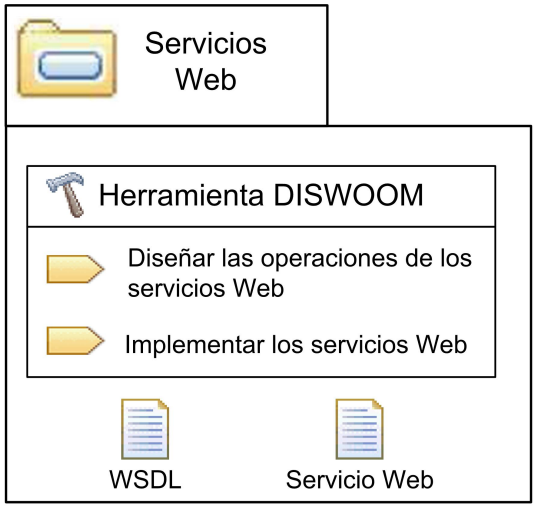

Figura 4.21: Actividades, Roles y Productos del *Diseño e Implementación de servicios Web*

4.5.1 Actividades

Esta etapa incluye actividades para el diseño e implementación de la funcionalidad definida en los modelos OO-Method / OOWS por medio de servicios Web. Para esto, se presenta un método que define e implementa de manera automática los servicios Web. Se distinguen dos actividades principales (ver figura 4.21):

1. *Diseño de las operaciones de los servicios Web*, a partir de los modelos OO-Method / OOWS definidos en las etapas de Elicitación de Requisitos (en el apartado 4.2) y Modelado Conceptual (en el apartado 4.3).
2. *Implementación* completa *de los servicios Web*, haciendo uso de la lógica generada por la etapa de Generación de Código (en el apartado 4.4).

El rol que lleva a cabo estas actividades es la herramienta que soporta la metodología propuesta en esta tesis. Esta herramienta, denominada `DISWOOM`

(Diseño e Implementación de Servicios Web aplicado a OO-Method/OOWS), se presenta en detalle en el capítulo 6.

Diseño de las operaciones de los servicios Web

La primera actividad (presentada en detalle en el capítulo 5) comienza tras la finalización de la etapa de Modelado. En esta actividad se estudian los diferentes modelos de los métodos OO-Method / OOWS para extraer la información necesaria para identificar las operaciones de los servicios Web (ver figuras 4.22 y 4.24).

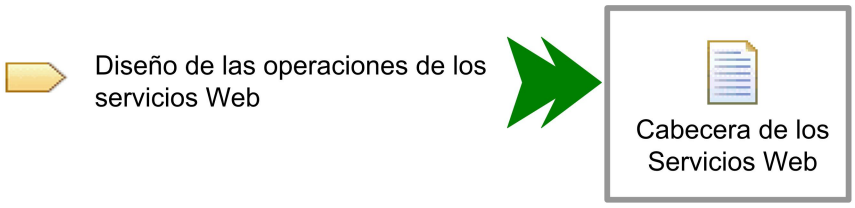

Figura 4.22: Producto del Diseño de las operaciones de los servicios Web

De cada una de las etapas presentadas, se utilizan un conjunto de modelos:

- *Especificación de Requisitos*: de esta etapa se utiliza el modelo de requisitos para determinar qué operaciones del sistema van a utilizar los usuarios potenciales.
  - El modelo de tareas proporciona información de las tareas (que se transformarán en operaciones) que puede desarrollar el usuario al interactuar con sistema. Para ello, tras estudiar las peculiaridades de este modelo, se detalla una estrategia para recorrerlo y seleccionar únicamente aquellas tareas que deben ser publicadas como operaciones por los servicios Web.
- *Modelado Conceptual*: de esta etapa se utilizan los modelos de OO-Method / OOWS para determinar qué operaciones de recuperación de información, de autenticación, de navegación y de presentación van a utilizar los usuarios potenciales.

- El modelo de usuarios ayuda a identificar si el sistema necesita operaciones para la autenticación y administración de usuarios.
- El modelo navegacional proporciona operaciones de recuperación de información. Además, y como una funcionalidad adicional a ofrecer por los servicio Web, este modelo proporciona información de la navegación que podría implementar una aplicación Web utilizando los servicios Web.
- El modelo de presentación ayuda a identificar qué presentación por defecto se ofrecerá a los usuarios si en lugar de utilizar las operaciones clásicas ofrecidas por los servicios Web utilizan las operaciones que además de devolver datos, devuelven estos preparados para ser mostrados directamente en una aplicación Web (más conocido como Web 2.0 [95, 96]).

Implementación de los servicios Web

La segunda actividad (comentada en detalle en el capítulo 6) comienza tras la identificación de las operaciones a incluir en los servicios Web. En esta actividad se estudian algunos modelos de los métodos OO-Method / OOWS para extraer la información necesaria para implementar las operaciones identificadas en la primera actividad. Estas operaciones se implementan a través del framework WIF de OOWS y del código generado por OlivaNOVA (ver figuras 4.23 y 4.24):

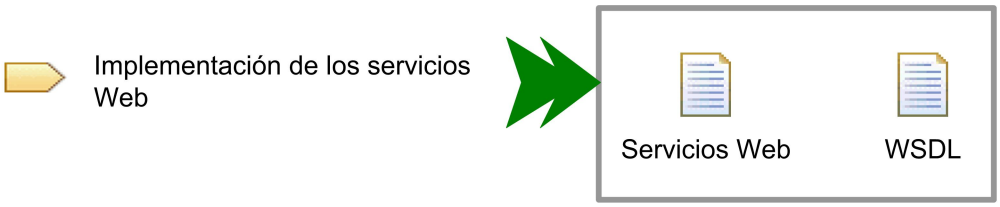

Figura 4.23: Productos de la Implementación de los servicios Web

- *Especificación de Requisitos*: de esta etapa se utiliza el modelo de requisitos para determinar cómo se implementan las operaciones identificadas del modelo de tareas.

  - La descripción de las tareas elementales proporciona información de qué operaciones implementan cada tarea.
    * Se recorre la descripción de la tarea para identificar qué operación corresponde con la tarea identificada en el modelo de tareas.
    * La plantilla de caracterización ofrece una descripción de la tarea/operación e información de qué roles de usuarios pueden realizarla.

- *Modelado Conceptual*: de esta etapa se utilizan los modelos de OO-Method / OOWS para determinar qué métodos del modelo implementan las operaciones identificadas en la actividad anterior.
  - El modelo de objetos se utiliza junto a la descripción de la tarea de la *Especificación de Requisitos* para identificar en qué clases están implementadas las operaciones identificadas del modelo de tareas.

- *Generación de Código*: de esta etapa se utiliza la implementación generada por OlivaNOVA para los modelos de OO-Method y la implementación del framework WIF, generada por OOWS Suite, para los modelos de OOWS.
  - De OOWS Suite se utiliza tanto la implementación de la capa de presentación como el framework WIF para acceder a OlivaNOVA.
    * Por un lado se utiliza el framework WIF para implementar de una forma sencilla las operaciones de OlivaNOVA.
    * Por otro lado, se utiliza la presentación definida para implementar las operaciones que implican presentación de información.
  - De OlivaNOVA se utilizan las capas de lógica de la aplicación y de persistencia para la implementación de las operaciones de los servicios Web.

4.5.2 Productos

Después de realizar estas dos actividades, se obtienen como productos el *código* y el *WSDL* que implementa las operaciones. Para ello se utiliza el código generado en la etapa de Generación de Código.

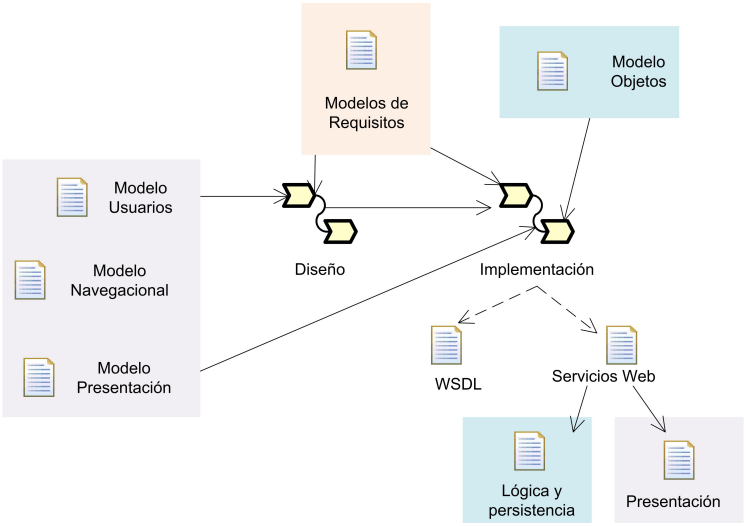

Figura 4.24: Etapa de diseño e implementación de servicios Web

Código

En la primera de las actividades, *diseño de las operaciones de los servicios Web*, se construye la cabecera de las operaciones de los servicios Web. Esta cabecera está compuesta por el nombre de la operación y los parámetros necesarios.

En la segunda actividad, *implementación de los servicios Web*, se implementa el cuerpo de las operaciones identificadas en la primera actividad. La implementación se realiza en Java. Aunque se ha elegido Java como lenguaje, las expresiones particulares de este lenguaje pueden adaptarse a cualquier otro lenguaje de programación que permita la implementación de servicios Web.

WSDL

Tras las dos actividades de esta etapa y junto al código generado, se genera además un fichero WSDL de las operaciones implementadas para su descubrimiento y utilización.

En los capítulos 5 y 6 se comenta en detalle cada uno de estos productos y la estrategia de *Modelo-A-Texto* seguido para obtenerlos.

4.6 Conclusiones

En este capítulo se ha presentado un método para el diseño e implementación de servicios Web, utilizando la notación y terminología propuesta en SPEM por la OMG. Esta notación permite representar las etapas, actividades, roles y productos involucrados en el método.

Se han presentado cada una de las etapas involucradas (Especificación de Requisitos, Modelado Conceptual y Generación de Código) introduciendo cada uno de los modelos (productos) que se van a ver en la tesis. Para cada una de las etapas se ha detallado cada actividad y producto utilizado en el desarrollo de esta tesis, detectándose posibles dependencias entre los distintos modelos que permiten indicar el orden de ejecución de cada una de las etapas.

Por último, se ha comentado brevemente la etapa de Diseño e Implementación de servicios Web, que se verá detalladamente en los siguientes capítulos (capítulos 5 y 6).

Los productos obtenidos de las etapas presentadas, se utilizan en capítulos posteriores para:

- Desarrollar una guía metodológica que permita el diseño e implementación de servicios Web.
- Desarrollar una herramienta que automatice el diseño e implementación de los servicios Web, basándose en los modelos (productos).
- Facilitar la construcción de la herramienta, proporcionando correspondencias precisas entre la generación automática que ofrece OO-Method / OOWS y la implementación de los servicios Web.

Capítulo 5

Diseño de servicios Web dirigido por modelos

En este capítulo se propone un método para el diseño de servicios Web a partir de los modelos conceptuales de OO-Method / OOWS. Este método permite identificar las operaciones de los servicios Web, una vez finalizada la etapa de modelado conceptual.

Las operaciones identificadas se pueden categorizar en distintos grupos funcionales. Esta categorización se realiza en base a la funcionalidad ofrecida por ellas. Cada grupo o conjunto de grupos funcionales será susceptible de formar un servicio Web.

Como se ha comentado en el capítulo 3, algunos de los métodos de la Ingeniería Web (Hera [59], OOHDM [29] y WebML [72]) incluyen el diseño de servicios Web en sus modelos conceptuales. Pero ninguno de ellos proporciona una guía para que el modelador diseñe de una forma manual o automática los servicios Web de los modelos conceptuales, sino que se deja de la mano del mismo tomar la decisión de qué operaciones ofrecer como servicios Web. En este capítulo se presenta un método para diseñar las operaciones.

El capítulo se estructura de la siguiente forma: en primer lugar, se introducen los servicios Web y los grupos funcionales que categorizan las operaciones de los servicios Web. A continuación, se presentan las operaciones que forman

cada uno de los grupos funcionales y cómo se identifican estas operaciones a partir de los modelos de OO-Method / OOWS. Con las conclusiones se finaliza el capítulo.

Para ejemplificar los diferentes subapartados, se aplica la propuesta al ejemplo del apéndice A. En este ejemplo se modela el caso de estudio de una aplicación de comercio electrónico como *Amazon*.

5.1 Servicios Web y Grupos Funcionales

En el capítulo 2 (página 18), se han introducido los conceptos básicos de los servicios Web: definición, características, escenarios y arquitectura. De los escenarios ahí presentados (ver apartado 2.3.3 en la página 20), esta tesis se centra en el escenario número 2 *"Implementar las aplicaciones como nuevos servicios Web"*, donde se diseñan e implementan nuevos servicios Web.

En [97] han identificado 12 propiedades deseables para el diseño de los servicios Web: descubrimiento, identificación, enlace, plataforma, interacción, orientado al interfaz, modelo, granularidad, estado, evolución, generación de código, conversación. De entre estas características, las únicas que dependen única y exclusivamente de la etapa de diseño, independientemente de la tecnología elegida para su implementación, son la granularidad [98] y la evolución.

En este trabajo de tesis, y a pesar de que es característica deseable en cualquier sistema, no se trata la característica de la evolución, sino que se plantea como un trabajo futuro. En cambio, la granularidad si que se tiene en cuenta a la hora de diseñar los servicios.

Durante los primeros años de desarrollo de los servicios Web, se creía que la forma más correcta de diseñar los servicios Web era hacer estos de grano grueso [40, 99] en lugar de los de grano fino. Tras varios años de análisis (IBM[1], [100]), se ha llegado al consenso de que la granularidad de los servicios Web debe ser óptima, es decir, manteniendo un balance entre grano fino y grano grueso.

[1]En http://www.ibm.com/developerworks/ se pueden encontrar numerosos artículos sobre este tema

En esta tesis, se van a diseñar los servicios Web procurando que su granularidad sea óptima. En algunos casos estos servicios serán de grano fino (por ejemplo `recuperarProducto`), y otras veces serán más cercanos al grano grueso (por ejemplo `presentacionDeInformacion`).

A continuación, se introducen los grupos funcionales donde se categorizan las operaciones de los servicios Web.

5.1.1 Grupos Funcionales

El método propuesto en esta tesis, identifica y clasifica las operaciones en un conjunto de grupos funcionales. Estos grupos funcionales ayudan a comprender la funcionalidad de las operaciones y permiten poder decidir si ofrecer un «grupo» de operaciones u otro, de forma sencilla.

En este apartado se presenta, a modo de introducción, los grupos funcionales (ver figura 5.1) que pueden determinar los servicios Web. Estos grupos funcionales conforman uno o varios servicios Web, y agrupan los distintos tipos de operaciones que se abordan en la fase de diseño e implementación tras la etapa de modelado conceptual.

Figura 5.1: Grupos Funcionales

Para representar tanto el conjunto de servicios (que en este caso es el mismo que el número de grupos funcionales) como las operaciones identificadas, para

cada uno de los servicios, se sigue la notación gráfica utilizada por Thomas Erl [101]. Los servicios Web son representados por bolas y el conjunto de servicios Web (repositorio) por una caja que los engloba (en la figura 5.1 se puede ver el repositorio de servicios Web que se obtiene en la tesis). Las operaciones de cada servicio se enumeran dentro de cada servicio (en la parte de abajo) separadas por una línea del nombre del servicio Web (en la figura 5.6 se puede ver el ejemplo del servicio Web *Lógica de la Aplicación* con sus operaciones).

Las operaciones identificadas de los modelos conceptuales se pueden categorizar en cinco grupos funcionales basándose en: (1) la funcionalidad ofrecida por cada operación y (2) en los modelos de los cuales se obtienen las operaciones. Los grupos funcionales son:

- **Lógica de la Aplicación**. Este grupo funcional está compuesto por operaciones que implementan los requisitos funcionales (lógica) de la aplicación. Las operaciones de este grupo se identifican del modelo de *Requisitos*. Por ejemplo, en el caso de estudio se obtienen operaciones como `anyadirProductoCestaCompra`, `modificarCestaCompra`, `borrarCestaCompra` o `realizarPago`.

- **Gestión de Usuarios**. Ofrece operaciones que implementan la autenticación, autorización y gestión de los usuarios. Las operaciones de este grupo se obtienen del modelo de *Usuarios*. Por ejemplo, en el caso de estudio se obtienen operaciones como `iniciarSesion`, `recordarContrasenya`, `crearUsuario` o `asignarUsuarioARol`.

- **Recuperación de Información**. Ofrece operaciones que implementan la búsqueda y recuperación de información. Las operaciones de este grupo se identifican de los *contextos navegacionales*, los cuales forman parte del modelo de *Navegación*. Por ejemplo, en el caso de estudio se obtienen operaciones como `recuperarProducto`, `indexarProductoIndex` o `buscarListaProductos`.

- **Soporte a la Navegación**. Ofrece operaciones que implementan la navegación permitida a cada uno de los usuarios del sistema. Las operaciones de este grupo se identifican de los *mapas navegacionales*, los cuales forman parte del modelo de *Navegación*. Este grupo complementa los servicios de los demás grupos funcionales, los cuales no tienen conocimiento alguno sobre la navegación permitida. Por ejemplo, en el caso de

estudio se obtienen operaciones como `recuperarEnlacesExploracion`, `recuperarEnlacesSecuencia` o `recuperarEnlacesOperacion`.

- **Soporte a la Presentación**. Ofrece operaciones que implementan un soporte que se asemeja al que se puede encontrar en los portlets [102] o en los mashups [103]. Las operaciones de este grupo se identifican de los modelos de *Navegación* y *Presentación*. Este grupo, al igual que el *Soporte a la Navegación*, complementa los servicios de los demás grupos funcionales que no tienen conocimiento alguno sobre la presentación final definida para el usuario. Por ejemplo, en el caso de estudio se obtienen operaciones como `presentacionDeInformacion`, `presentacionDeProducto` o `presentacionDeComentariosClientes`.

En los siguientes apartados se detalla cada uno de los grupos funcionales. La estructura seguida para cada grupo funcional es la siguiente: primero se identifica el modelo origen desde el que se diseñan las operaciones. A continuación, se presentan una serie de reglas con los pasos a seguir para el diseño de las operaciones a partir del modelo origen. Estas reglas se implementan posteriormente, en una serie de algoritmos en el lenguaje de *Modelo-A-Texto* MOFScript. Finalmente, se ponen en prácticas las ideas presentadas para el caso de estudio del comercio electrónico (presentado en el anexo A).

5.2 Lógica de la Aplicación

El grupo funcional de *Lógica de la Aplicación* está formado por operaciones que implementan los requisitos funcionales del usuario. En este grupo se identifican las operaciones a partir de la lógica definida en los modelos, que se corresponde con la especificación del sistema a desarrollar.

Las operaciones de este grupo dependen de la aplicación a desarrollar, y en concreto de los modelos del dominio construidos por el modelador. El trabajo desarrollado en la tesis propone una serie de reglas que permiten identificar las operaciones a partir del modelo de *Requisitos* definido en OO-Method / OOWS (ver apartado 4.2 en la página 60).

Los elementos que forman parte del modelo de requisitos y permiten iden-

tificar las operaciones de este grupo funcional son:

- Un *modelo de tareas*, que permite identificar las operaciones del servicio Web.
- Una *descripción de la tarea* para cada una de las tareas hoja, que permite identificar los parámetros y el método del modelo de objetos que implementa la operación identificada.
- Una *plantilla de caracterización* para cada una de las tareas hoja, que permite identificar la descripción de la operación y los usuarios que pueden ejecutarla.

En el apartado 4.2, se puede encontrar información más detallada sobre cada uno de estos elementos del modelo de requisitos.

5.2.1 Reglas de transformación

En este apartado se presentan cuatro reglas para el diseño de las operaciones de la *Lógica de la Aplicación*. Para presentar las cuatro reglas se ha dividido el apartado en tres partes, según el elemento del modelo de requisitos que se utiliza para definir cada regla: primero el *modelo de tarea*, a continuación la *descripción* asociada a cada tarea y finalmente la *plantilla de caracterización*. Para finalizar el apartado, se muestran los dos algoritmos que implementan el diseño de las operaciones.

Modelo de tareas

Las operaciones de este grupo funcional, se diseñan a partir de las tareas que forman el modelo de tareas. Como se ha comentado anteriormente en el apartado 4.2.2, un modelo de tareas identifica los objetivos del usuario, qué espera poder realizar a través de la aplicación. Por lo tanto, este modelo ayuda a identificar las operaciones que debe publicar un servicio Web para dar soporte a las tareas que desea realizar el usuario.

A continuación, se presenta cómo recorrer el modelo de tareas y cómo identificar las tareas que van a originar las operaciones del servicio Web. Es importante que se recorra el modelo pasando por todas las tareas y sin repetir ninguna.

El modelo de tareas tiene estructura de árbol. La forma de realizar el recorrido está relacionada con el orden en que se visitan los elementos del árbol, puesto que lo único que se debe garantizar es que se pase exactamente una sola vez por cada una de las tareas (que son los nodos del árbol). El recorrido en *preorden* [104] primero procesa la tarea y luego, si no ha sido seleccionada, recorre el subárbol de dicha tarea. Este recorrido garantiza que se pasa sólo una vez por cada tarea y además también garantiza que una vez seleccionada una tarea como posible operación a publicar, ya no se procesan el resto de tareas que hay por debajo.

Por lo tanto, siguiendo las premisas del recorrido en preorden, se comienza el recorrido del modelo de tareas en la tarea raíz, recorriendo la tarea de la izquierda y después la tarea de la derecha. Por ejemplo, en la figura 5.2 se «visita» primero la tarea `Realizar Compra`, después `Comprar Productos` seguido de `Llenar Cesta Compra`, `Seleccionar Producto`, etc.

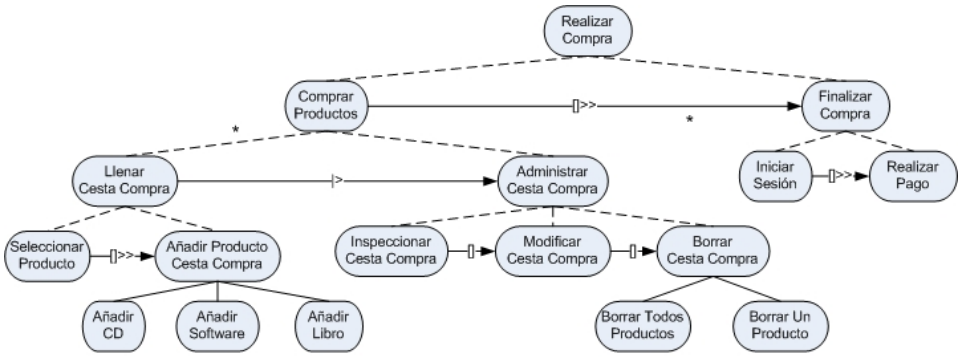

Figura 5.2: Modelo de Tareas

Mientras se recorre el modelo, se siguen las siguientes premisas para seleccionar las tareas que van a convertirse en operaciones candidatas a ser publicadas:

1. La relación estructural, representada por líneas continuas entre tareas,

muestra tareas que se descomponen en tareas más simples. Cada una de estas tareas, se puede llevar a cabo independientemente una de otra. Por tanto, la granularidad de estas tareas es demasiado pequeña como para ser publicada y la tarea que contiene a varias tareas tendrá una granularidad más óptima para ser publicada.

2. Una tarea hoja es una tarea que no se descompone en subtareas, por lo tanto, si se ha llegado hasta un tarea hoja, significa que se habrán analizado todas las tareas relevantes de la rama.

3. La primera tarea de una relación temporal, representada por líneas discontinuas continuas entre tareas, no será seleccionada como operación a publicar por el servicio Web si es una tarea hoja. Esto se debe a que realmente este tipo de tareas no corresponden con una operación de interacción entre el sistema y el usuario, sino de una acción a llevar a cabo por el usuario.

Cuando se selecciona una tarea, se sigue recorriendo la parte del modelo que comienza por la subtarea derecha de la seleccionada. Por ejemplo, si se selecciona la tarea `Modificar Cesta Compra`, entonces se sigue por la tarea `Borrar Cesta Compra` para recorrer ese subárbol.

Siguiendo estas premisas, mientras se recorre el modelo de tareas se buscan las tareas candidatas a ser publicadas como operaciones por el servicio Web. Si la relación en la que participan es una relación estructural se sigue la siguiente regla:

REGLA RE: Refinamiento Estructural

Entrada: Modelo de tareas

Dada una relación estructural, se obtiene como operación a publicar la tarea inmediatamente superior a la relación. Esta regla aplica la primera premisa propuesta.

Salida: Lista de operaciones del tipo *`nombreTarea`*

En el ejemplo del caso de estudio del comercio electrónico, siguiendo esta regla, se seleccionan las tareas `Añadir Producto Cesta Compra` y `Borrar Cesta Compra` como operaciones a publicar.

Si al recorrer el modelo de tareas, la relación en la que participan las tareas es una relación temporal, se sigue la siguiente regla:

REGLA RT: Refinamiento Temporal

Entrada: Modelo de tareas

Dada una relación temporal, se obtienen aquellas tareas hoja que no participan en una relación estructural. Esta regla aplica la segunda premisa.

Salida: Lista de operaciones del tipo *nombreTarea*

La *regla RT* posee como excepción (premisa 3) que la primera tarea de la relación siempre será excluida como operación a publicar.

En el ejemplo del caso de estudio, siguiendo esta regla, se seleccionan las tareas `Modificar Cesta Compra` y `Realizar Pago` como operaciones a publicar.

Para que el diseño de servicios Web esté completo, las operaciones deben tener definidos los parámetros de entrada y de salida. Cada tarea que ha servido para identificar las operaciones (siguiendo las reglas RE y RT) tiene asociado una descripción de la tarea y otra de datos (como se ha comentado en 4.2). A partir de estas descripciones, y junto al modelo de clases, se pueden derivar los parámetros que completan el diseño de las operaciones.

Asociado a cada tarea hoja del modelo de tareas, en el modelo de requisitos se define:

1. una *descripción de la tarea* (ver apartado 4.2.2 en la página 62) mediante diagramas de actividad, que indica las acciones que el usuario debe realizar en el sistema para llevar a cabo la tarea. La descripción de la tarea permite definir los parámetros de las operaciones (ver el apartado 5.2.1).

2. una *plantilla de caracterización* de la tarea, que indica:

   - los *objetivos* que se deben alcanzar al realizar la tarea (*Goals*)
   - los *usuarios* que pueden llevar a cabo la tarea (*Users*)

   La plantilla de caracterización permite definir otros elementos de las operaciones como son la descripción de las acciones que lleva a cabo la operación y los permisos de la misma (ver el apartado 5.2.1).

Descripción de las tareas

En la descripción de las tareas (ver apartado 4.2.2, en la página 61), cada nodo del diagrama de actividad constituye:

1. Una acción del sistema, representada mediante nodos marcados con los estereotipos «`function`» o «`search`».

2. Un punto de interacción (*PI*), representado mediante nodos marcados con los estereotipos «`input`» o «`output`».

Un *PI* representa cada interacción entre el sistema y el usuario durante la ejecución de una tarea, con el fin de que el usuario introduzca información («`input`») o que el sistema proporcione al usuario información o acceso a una determinada funcionalidad («`output`»). En este caso, el nombre asociado al nodo indica la entidad (clase del modelo de clases) del sistema con la que está relacionada la información intercambiada entre el usuario y el sistema.

En el contexto de la tesis, las acciones del sistema que se llevan a cabo tras un *PI* estereotipado con «`output`», permiten derivar el/los métodos de las clase/s que implementan cada operación del servicio Web (ver figura 5.3). Las acciones del sistema definen la funcionalidad de la operación del servicio Web identificado en esta tarea. La clase a la que pertenece esta operación viene definida por la entidad asociada al *PI* y será uno de los argumentos de la operación del servicio Web. Esto se resume en la siguiente regla:

REGLA IDC: Identificación a partir del Modelo de Clases

Entrada: Descripción de la tarea

Para identificar la clase que implementa el *PI*, es necesaria la descripción de la tarea junto con el modelo de clases. Se detecta la clase participante en una operación comparando la entidad especificada en el nodo de la descripción con el modelo de clases.

Regla IDCa: Las tareas identificadas mediante la *regla RE* no son tareas hoja y por lo tanto no tienen descripción asociada. Para este caso se recorre la descripción de la primera tarea hija (que sí que es hoja) y de esta forma poder obtener los parámetros de la operación.

Regla IDCb: Las tareas identificadas mediante la *regla RT* son tareas hoja y por lo tanto se recorre su descripción para obtener los parámetros de la operación.

Salida: Parámetros de la operación

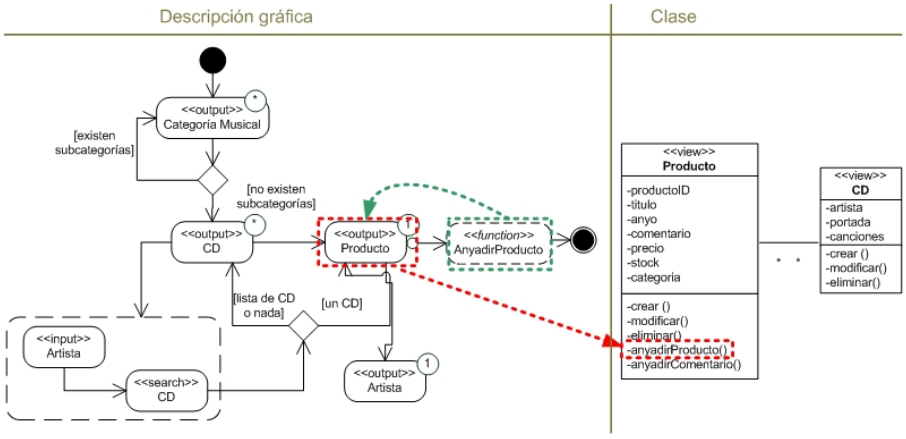

Figura 5.3: Descripción de la tarea `Add CD`

En la figura 5.3 se muestra la descripción de la tarea `Añadir CD`, que siguiendo la *regla RE* ha sido seleccionada como operación. En esta figura, siguiendo la *regla IDC* (más concretamente la *regla IDCa*), se puede ver la

acción del sistema `AnyadirProducto`, precedida por un *PI* estereotipado con «`output`» asociado a la entidad `Producto`. Esto indica que la clase `Producto` contendrá un método llamado `AnyadirProducto`, el cual implementa la operación identificada (en esta caso es la tarea superior `anyadirProductoCestaCompra`).

Plantilla de caracterización

La plantilla de caracterización de la tarea (ver apartado 4.2.2, en la página 61), proporciona información relevante sobre (1) qué hace la operación y (2) los permisos de la operación del servicio Web.

La plantilla de caracterización está formada por tres propiedades:

- *Tarea*: indica a qué tarea pertenece la plantilla de caracterización.
- *Objetivo*: es donde se define la descripción de la operación.
- *Usuarios*: es donde se definen los roles de usuarios que pueden invocar la operación.

Por lo tanto, la siguiente regla utiliza esta plantilla para:

REGLA DO: Descripción de la Operación

Entrada: Plantilla de caracterización

Para identificar la descripción de la operación y qué usuarios pueden invocarla, es necesaria la plantilla de caracterización de la tarea identificada:

Regla DOa: Las tareas identificadas siguiendo la *regla RE* no son tareas hoja y por lo tanto no tienen plantilla de caracterización. Para este caso se recorre la plantilla de caracterización de la primera tarea hija (que sí que es hoja) para poder obtener la descripción y los usuarios de la operación.

Regla DOb: Las tareas identificadas mediante la *regla RT* son tareas hoja y por lo tanto se recorre su plantilla de caracterización para obtener la descripción y los usuarios de la operación.

Salida: Descripción de la operación y usuarios que la pueden invocar

En la figura 5.4 se muestra la plantilla de caracterización de la tarea `Añadir CD`. Siguiendo la *regla DO* (más concretamente la *regla DOa*), esta caracterización pertenece a la tarea `Añadir CD`, la primera tarea hija de la tarea `Añadir Producto`, que siguiendo la *regla RE* ha sido seleccionada como operación. La descripción de la tarea, y por tanto de la operación, es "El usuario añade un producto a su carrito de la compra". Los usuarios que pueden invocarla son los usuarios de los roles `Visitante` y `Cliente`.

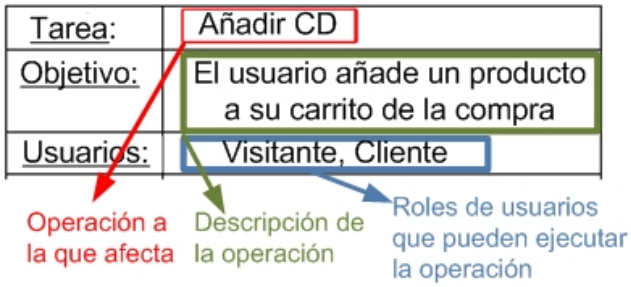

Figura 5.4: Plantilla de caracterización de la tarea `Añadir CD`

En resumen, la *regla RE* y la *regla RT* se corresponden a la identificación de las operaciones, la *regla IDC* define los parámetros de entrada y/o salida de las operaciones y la *regla DO* se corresponden con la descripción y seguridad a aplicar en cada operación.

A continuación se presenta la implementación de estas reglas utilizando el lenguaje de transformación *Modelo-A-Texto* MOFScript.

Implementación de las reglas

Los algoritmos que se presentan a continuación, están formados por: una *entrada* que indica qué espera el algoritmo que se le pase como entrada, el modelo

del cuál se identifican las operaciones; y finalmente la *nomenclatura*, que son los elementos del modelo de entrada que se van a utilizar durante el algoritmo.

- Entrada:
  - Modelo de Tareas (`dt`) del Modelo de Requisitos
  - Modelo de Clases (`dc`) del Modelo Conceptual
- Nomenclatura:
  - `dt.Task, t`: la tarea actual
  - `t.nombre`: el nombre de la tarea
  - `t.StructuralRefinement`: refinamiento estructural de la tarea
  - `t.TemporalRefinement`: refinamiento temporal de la tarea
  - `dc.Class, c`: clase del modelo de clases
  - `c.Operation, o`: operación de la clase

El primer algoritmo implementa la *regla RE*, que se corresponde con las tareas de refinamiento estructural. Además, se implementa la *regla IDC* para definir los parámetros de entrada y/o salida de las operaciones y la *regla DO* con la descripción y seguridad a aplicar en cada operación.

Algoritmo 1 Refinamiento Estructural

```
self.Task->forEach(t:tasksRequirements.Task) {
 if(not t.StructuralRefinement.isEmpty()) {  //Refinamiento Estructural
     t.StructuralRefinement->forEach(sr:tasksRequirements.
       StructuralRefinement) {
       sr.StructuralRefinementTo->forEach(tsr:tasksRequirements.Task) {
         if((tsr.StructuralRefinement.isEmpty()) and
           (tsr.TemporalRefinement.isEmpty())) {
           ls_nombreOperacion = t.name.firstToLower().replace(" ", "")
           dc.Class->forEach(c:oows.Class){
             c.Operation->forEach(o:oows.Operation |
               o.ref.equals(ls_nombreOperacion)){
               ls_clase = c.id
               ls_metodo = o.id }  }  }  } } } //Refinamiento Estructural
```

Como se puede observar en el algoritmo 1, se recorre el árbol de tareas visitando sólo aquellas tareas que participan en una relación de tipo estructural (*regla RE*). Para cada una de estas tareas: (1) se toma como nombre de la operación el nombre de la tarea; (2) se identifican los parámetros de la operación y dónde se implementa a partir de la operación de la clase representada en el modelo de clases (*regla IDCa*).

El segundo algoritmo implementa la *regla RT* que se corresponde con las tareas de refinamiento temporal. Al igual que en el primer algoritmo, se implementa la *regla IDC* para definir los parámetros de entrada y/o salida de las operaciones y la *regla DO* con la descripción y seguridad a aplicar en cada operación.

Algoritmo 2 Refinamiento Temporal

```
if (not t.TemporalRefinement.isEmpty()) {//Refinamiento Temporal
  t.TemporalRefinement->forEach(tr:tasksRequirements.TemporalRefinement) {
    if ((tr.TemporalRefinementTo.StructuralRefinement.isEmpty()) and
      (tr.TemporalRefinementTo.TemporalRefinement.isEmpty())) {
      ll_posiblesTareas->forEach(pt) {
        if (pt == tr.TemporalRefinementTo.name) {
          ls_nombreOperacion = tr.TemporalRefinementTo.name.
              firstToLower().replace(" ", "")
          dc.Class->forEach(c:oows.Class){
            c.Operation->forEach(o:oows.Operation |
              o.ref.equals(ls_nombreOperacion)){
              ls_clase = c.id
              ls_metodo = o.id } } } } } }//Refinamiento Temporal
```

Como se puede observar en el algoritmo 2, se recorre el árbol de tareas visitando sólo aquellas tareas que participan en una relación de tipo temporal (*regla RT*). Para cada una de estas tareas: (1) se toma como nombre de la operación el nombre de la tarea, excluyendo la primera tarea de la relación; (2) se identifican los parámetros de la operación y dónde se implementa a partir de la operación de la clase representada en el modelo de clases (*regla IDCb*).

5.2.2 Ejemplo del caso de estudio

A continuación, se detalla cada una de las operaciones detectadas siguiendo los pasos del algoritmo para el caso de estudio. Los elementos del caso de estudio sobre los que se aplica este algoritmo son los modelos de tareas de cada uno de los roles de usuarios de la aplicación.

En la figura 5.5 se muestran las operaciones y las tareas desde las que han sido identificadas tras realizar la traza de los algoritmos 1 y 2 sobre el modelo para los usuarios del rol `Cliente`.

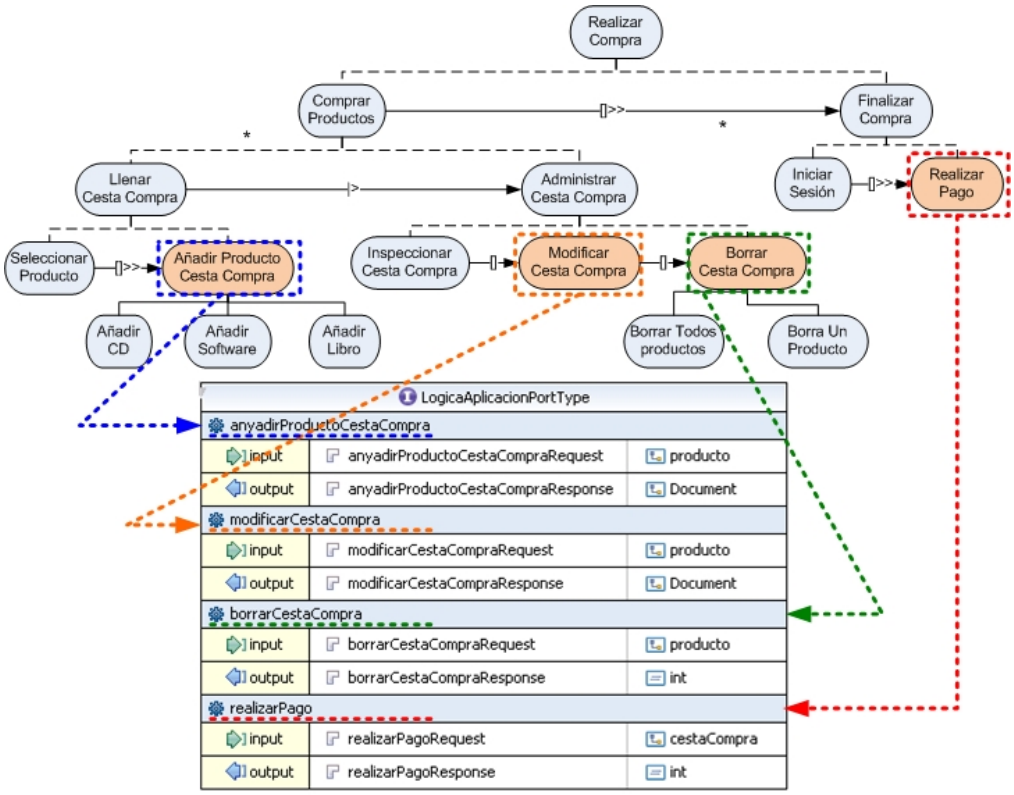

Figura 5.5: Identificación de operaciones para la *Lógica de la Aplicación*

- `anyadirProductoCestaCompra`
  - Descripción: El usuario añade un producto a su carrito de la compra.
  - Parámetros de entrada: Producto que se quiere añadir a la cesta.
  - Parámetros de salida: La cesta con el producto añadido.
  - Usuarios: Visitante y Cliente.
- `modificarCestaCompra`
  - Descripción: esta operación permite modificar un producto que se encuentra en la cesta de la compra.
  - Parámetros de entrada: Producto de la cesta que se desea modificar.
  - Parámetros de salida: La cesta con el producto modificado.
  - Usuarios: Visitante y Cliente.
- `borrarCestaCompra`
  - Descripción: esta operación permite eliminar un producto o todos los productos que se encuentran en la cesta.
  - Parámetros de entrada: Opcionalmente el producto que se desea eliminar o ninguno si se desea vaciar por completo la cesta de la compra.
  - Parámetros de salida: 0 si ha ido todo bien o -1 en caso contrario.
  - Usuarios: Visitante y Cliente.
- `realizarPago`
  - Descripción: esta operación permite administrar el pago de las compras realizadas.
  - Parámetros de entrada: cesta de la compra que se desea pagar.
  - Parámetros de salida: 0 si ha ido todo bien o -1 en caso contrario.
  - Usuarios: Visitante y Cliente.

De las operaciones identificadas el modelador/diseñador debe de seleccionar aquellas operaciones que desea publicar. Para una aplicación modelada con OO-Method / OOWS, por defecto se publican todas las operaciones identificadas. En la figura 5.6 se muestran las operaciones publicadas para el grupo funcional *Lógica de la Aplicación*.

Figura 5.6: Operaciones publicadas de la *Lógica de la Aplicación*

5.3 Gestión de Usuarios

Este grupo funcional está formado por operaciones que implementan la autenticación, autorización y gestión de los usuarios, necesarias para interactuar con los servicios Web.

Las operaciones de este grupo no dependen de la aplicación a desarrollar, sino que dependen de la existencia de usuarios registrados que desean acceder al sistema. El trabajo desarrollado en la tesis propone una serie de reglas que permiten identificar las operaciones a partir del modelo de *Usuarios* del método OO-Method/OOWS (ver apartado 4.3.2 en la página 67).

El elemento que forma parte del modelo y permite identificar las operaciones de este grupo funcional es el *modelo de usuarios*, que proporciona información sobre los tipos de usuarios (roles) existentes en la aplicación y les asigna permisos para interactuar con el sistema, proveyendo un control basado en roles (RBAC [47], Role-Based Access Control).

RBAC es un estándar para el control de acceso basado en roles (ver apartado 2.4 en la página 29) que proporciona guías para llevar a cabo el control de acceso en una aplicación. La propuesta que se desarrolla para este grupo funcional, es un proceso de abstracción del modelo de RBAC, simplificando su modelo de referencia y extendiendo las operaciones de gestión de identidad y control de acceso basado en roles.

La noción básica de RBAC es que tanto los permisos como los usuarios

se asignan a roles. Según el significado que da RBAC a los elementos básicos de su modelo, podemos crear una equivalencia entre sus elementos, los que conforman OO-Method / OOWS y los servicios Web (ver la tabla 5.1).

Tabla 5.1: Correspondencia entre los elementos de RBAC y los elementos de la tesis

Elemento de RBAC	Gestión de Usuarios
Objeto	Operaciones de los servicios Web
Operación	-
Permiso	Permiso o denegación de acceso
Rol	Roles de OO-Method / OOWS
Usuario	Usuarios que forman parte de los roles

- Un *Objeto* puede ser cualquier recurso del sistema que necesite control de acceso. En los servicios Web, las operaciones son los recursos que necesitan control de acceso.

- Una *Operación* es una funcionalidad ejecutable por el usuario. Con esta definición se refiere a qué tipo de funcionalidad se puede realizar sobre los objetos. En el caso de los servicios Web, la única operación definida es si se tiene acceso o no a los objetos (en el caso de la tesis son las operaciones del servicio Web). Por lo tanto, este elemento se puede suprimir sin que el resto de elementos sufran cambios.

- Un *Permiso* es el consentimiento de llevar a cabo una operación sobre alguno de los objetos RBAC. En el caso de los servicios Web, al no tener operaciones se simplifica a tener o no permiso para acceder al objeto.

- Un *Rol* es una función dentro de una organización. Este elemento de RBAC se corresponde con lo que en OO-Method / OOWS se llama también *Rol*.

- Un *Usuario* es una persona que puede acceder al sistema. Este elemento se corresponde con cada usuario que forma parte de un rol.

Por lo tanto, el modelo RBAC se simplifica y queda como muestra la figura 5.7. La diferencia entre el modelo RBAC completo (figura 2.12, página 30) y el modelo de RBAC simplificado está en que para el modelo de RBAC simplificado los permisos se asignan directamente a los objetos, puesto que desaparecen las operaciones tal y como se ha comentado anteriormente. De esta forma se simplifica notablemente la implementación.

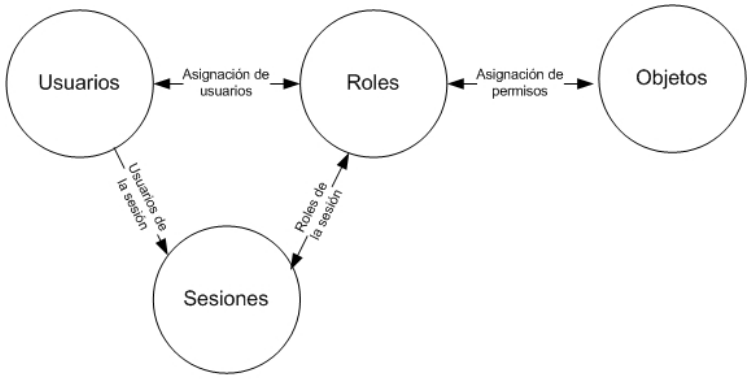

Figura 5.7: Modelo RBAC simplificado

5.3.1 Reglas de transformación

En este apartado se presenta una regla para el diseño de las operaciones de la *Gestión de Usuarios*. A continuación, se presentan las operaciones derivadas del modelo RBAC que se utilizan, y las operaciones que extienden el modelo. Para finalizar el apartado, se muestra el algoritmo que implementa la regla para el diseño de las operaciones.

Modelo de Usuarios

Al igual que el modelo de tareas del apartado anterior, el modelo de usuarios tiene estructura de árbol. La forma de recorrerlo es igual a la presentada en el apartado anterior y se va a realizar en *preorden*.

Mientras se recorre el modelo de usuarios, se buscan roles que cumplan la siguiente regla:

REGLA UR: Usuario Registrado

Entrada: Modelo de Usuarios

Recorrer el modelo de usuarios hasta encontrar un usuario registrado. Si se encuentra, se publican todas las operaciones de este grupo, sino no se publica ninguna.

Salida: Lista de operaciones del tipo `gestionUsuarios`

En el ejemplo del caso de estudio del comercio electrónico, siguiendo esta regla, se encuentra al rol `Cliente` que se corresponde con un usuario registrado (representado gráficamente con un candado en la cabeza) y por lo tanto se diseñan todas las operaciones de este grupo funcional.

Modelo de RBAC

La especificación funcional de RBAC [47] (ver el apartado 2.4 en la página 29) da soporte funcional al modelo de referencia y propone una serie de operaciones necesarias en cualquier sistema que trabaje con Control de Acceso Basado en Roles. Como se ha comentado en el apartado anterior, en esta tesis se utiliza un modelo simplificado de RBAC y por lo tanto se puede utilizar un subconjunto de las operaciones propuestas por RBAC.

A la especificación funcional de RBAC han de sumarse las operaciones identificadas en los trabajos de investigación realizados por la autora de la tesis (ver [105, 106, 107]). De esta forma se extiende la especificación funcional de RBAC para cubrir las necesidades de las aplicaciones y servicios Web.

Primero, se van a ver y categorizar las operaciones presentadas en la especificación funcional de RBAC para a continuación presentar las operaciones identificadas en los trabajos de investigación.

A continuación se presentan las operaciones de la especificación funcional de *Core RBAC (CRBAC)*[47] que se utilizan, junto con la nomenclatura seguida en la tesis:

- De la administración de elementos, se van a utilizar las siguientes operaciones:
  - `AddUser`, renombrada a `crearUsuario`
  - `DeleteUser`, renombrada a `eliminarUsuario`
  - `AddRole`, renombrada a `crearRol`
  - `DeleteRole`, renombrada a `eliminarRol`
  - `AssignUser`, renombrada a `asignarUsuarioARol`
  - `DeassignUser`, renombrada a `desasignarUsuarioDeRol`
  - `GrantPermission`, renombrada a `asignarPermisoARol`
  - `RevokePermission`, renombrada a `desasignarPermisoDeRol`
- Del soporte al sistema para CRBAC, se van a utilizar las siguientes operaciones:
  - `CreateSession`, renombrada a `iniciarSesion`
  - `DeleteSession`, renombrada a `cerrarSesion`

  Las operaciones `CheckAccess`, `AddActiveRole` y `DropActiveRole` no se utilizan en la tesis dado que los roles que se van a asignar a una sesión van a ser siempre los roles del usuario.
- De la consulta para CRBAC, se van a utilizar las siguientes operaciones:
  - `AssignedUsers`, renombrada a `buscarUsuariosDeRol`
  - `AssignedRoles`, renombrada a `buscarRolesDeUsuario`
- Por último, del soporte de funciones avanzadas, se van a utilizar:
  - `RolePermissions`, renombrada a `buscarPermisosDeRol`

  Las operaciones `UserPermissions`, `RoleOperationsOnObject` y `UserOperationsOnObject` son operaciones que no van a ser utilizadas en la tesis por no existir en el modelo simplificado de CRBAC. Además, cabe comentar que las operaciones presentadas no tienen en cuenta la variable `operación` de RBAC.

En este conjunto de operaciones sólo están definidas operaciones de creación, borrado y algunas de recuperación. A este grupo le faltan operaciones que permitan modificar los datos existentes tanto de la información de los usuarios como de los permisos u objetos. Además, hacen faltan operaciones para la identificación de usuarios (para poder interactuar con el sistema) y para la administración de objetos susceptibles de tener control de acceso. Las operaciones que extienden la funcionalidad de RBAC son:

- Modificación de datos: `modificarUsuario` y `modificarRol`.
- Identificación de usuarios: `iniciarSesion`, `cerrarSesion` y `recordarContrasenya`.

Resumiendo, las operaciones identificadas del diagrama de usuarios basándose en el modelo CRBAC se clasifican en cuatro tipos (ver figura 5.8) y se presentan a continuación.

Figura 5.8: Clasificación de las operaciones de *Gestión de Usuarios*

1. **Administración de Sesiones**: Operaciones que proveen soporte a la identificación de usuarios (ver figura 5.9). Se encargan de crear y eliminar la sesión de un usuario. Adicionalmente, se encargan de recordar la contraseña. Las operaciones que lo implementan son: `iniciarSesion`, `cerrarSesion`, `recordarContrasenya`
2. **Administración de Usuarios**: Operaciones que dan soporte a la administración de usuarios (ver figura 5.9). Se encargan de crear y eliminar un usuario, y modificar sus datos o su contraseña. Las operaciones que lo implementan son: `crearUsuario`, `eliminarUsuario`, `modificarUsuario`, `modificarContrasenya`

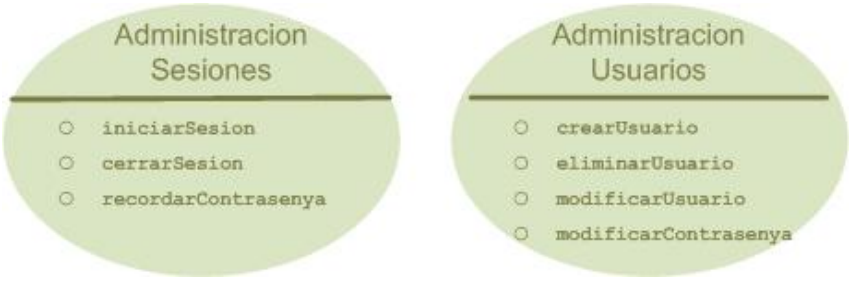

Figura 5.9: Operaciones de administración de sesiones y de usuarios

3. **Administración de Roles**: Operaciones que dan soporte a la administración de roles (ver figura 5.10). Se encargan de crear y eliminar los roles del sistema, asignar y desasignar usuarios a los roles y de realizar diversas búsquedas. Las operaciones que lo implementan son: `crearRol`, `eliminarRol`, `modificarRol`, `asignarUsuarioARol`, `desasignarUsuarioDeRol`, `buscarUsuariosDeRol`, `buscarRolesDeUsuario`

4. **Administración de Permisos**: Operaciones que dan soporte a la administración de permisos (ver figura 5.10). Se encargan de asignar y desasignar permisos a los roles y de realizar diversas búsquedas. Las operaciones que lo implementan son: `asignarPermisosARol`, `desasignarPermisosDeRol`, `buscarPermisosDeSesion`

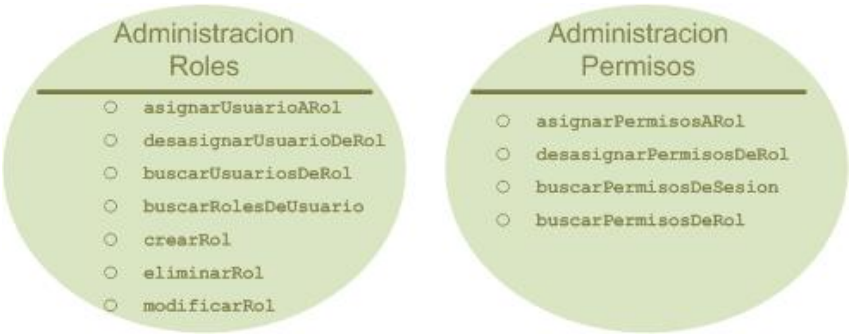

Figura 5.10: Operaciones de administración de roles y de permisos

A continuación, se presenta la implementación de la *regla UR* utilizando el lenguaje de transformación *Modelo-A-Texto* MOFScript.

Implementación de las reglas

La estructura del algoritmo que se presenta a continuación está formada por: una *entrada* que indica qué espera el algoritmo que se le pase como entrada, el modelo del cuál se identifican las operaciones; la *nomenclatura*, que son los elementos del modelo de entrada que se van a utilizar durante el algoritmo; y finalmente las *funciones*, que son operaciones que realizan una acción en concreto y ayudan a simplificar el funcionamiento del algoritmo.

- Entrada:
  - Modelo de Usuarios (`MU`)
- Nomenclatura:
  - `MU.ur`: el rol actual del modelo de usuarios
- Funciones:
  - `construirSesiones`: diseña las operaciones del tipo *Administración de Sesiones*, en la página 107.
  - `construirUsuarios`: diseña las operaciones del tipo *Administración de Usuarios*, en la página 107.
  - `construirRoles`: diseña las operaciones del tipo *Administración de Roles*, en la página 108.
  - `construirPermisos`: diseña las operaciones del tipo *Administración de Permisos*, en la página 108.

Algoritmo 3 Usuarios Registrados

```
self.NavigationalModel.UserRol -> forEach(ur:oows.UserRol){
  if(ur.registered){
     ur.construirSesiones()
     ur.construirUsuarios()
     ur.construirRoles()
     ur.construirPermisos() } }
```

En el algoritmo 3, se observan las llamadas que se realizan para la implementación de las distintas operaciones.

5.3.2 Ejemplo del caso de estudio

A continuación, se detalla cada una de las operaciones del caso de estudio. En la figura 5.11 se muestran las operaciones y los usuarios desde las que han sido identificadas siguiendo el algoritmo 3.

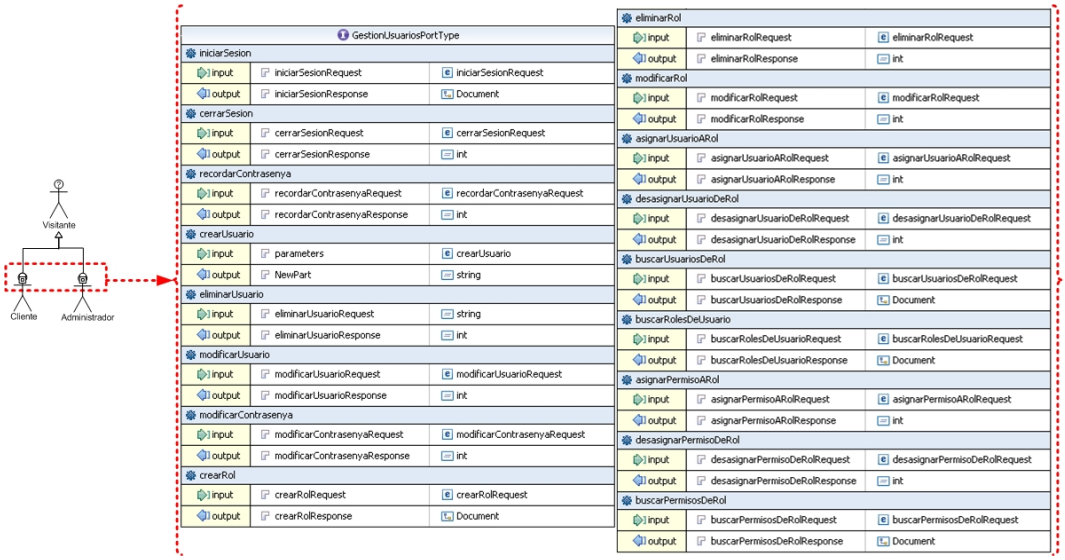

Figura 5.11: Identificación de operaciones para la *Gestión de Usuarios*

Administración de Sesiones

La especificación de las operaciones encargadas de la administración de sesiones es:

- `iniciarSesion`
  - Descripción: esta operación crea una nueva sesión con el usuario y un grupo de roles activos.
  - Parámetros de entrada: Datos del inicio de sesión (`usuario` y `contrasenya`).

  - Parámetros de salida: Si los datos de entrada se corresponden con un usuario, se crea una sesión en el sistema y se devuelve un hash `sesion` que identifica dicha sesión.

- `cerrarSesion`
  - Descripción: esta operación elimina la sesión del usuario.
  - Parámetros de entrada: `sesion`.
  - Parámetros de salida: 0 si ha ido todo bien o -1 en caso contrario.

- `recordarContrasenya`
  - Descripción: esta operación envía un correo al usuario para recordarle su contraseña si contesta correctamente a la pregunta secreta.
  - Parámetros de entrada: `usuario` y `respuesta`.
  - Parámetros de salida: correo electrónico al usuario recordándole su contraseña.

Administración de Usuarios

La especificación de las operaciones encargadas de la administración de usuarios es:

- `crearUsuario`
  - Descripción: esta operación crea un nuevo usuario.
  - Parámetros de entrada: Datos del `usuario`.
  - Parámetros de salida: 0 si ha ido todo bien o -1 en caso contrario.

- `eliminarUsuario`
  - Descripción: esta operación elimina un usuario existente de la base de datos del sistema.
  - Parámetros de entrada: Identificador del `usuario`.
  - Parámetros de salida: 0 si ha ido todo bien o -1 en caso contrario.

- `modificarUsuario`
  - Descripción: esta operación modifica los datos de un usuario.
  - Parámetros de entrada: Datos del `usuario`.
  - Parámetros de salida: 0 si ha ido todo bien o -1 en caso contrario.
- `modificarContrasenya`
  - Descripción: modifica la contraseña de un usuario.
  - Parámetros de entrada: Identificador del `usuario`, `contrasenya actual` y la `nueva contrasenya`.
  - Parámetros de salida: 0 si ha ido todo bien o -1 en caso contrario.

Administración de Roles

La especificación de las operaciones encargadas de la administración de roles son:

- `crearRol`
  - Descripción: esta operación crea un nuevo rol.
  - Parámetros de entrada: Datos del `rol`
  - Parámetros de salida: 0 si ha ido todo bien o -1 en caso contrario.
- `eliminarRol`
  - Descripción: esta operación elimina un rol existente de la base de datos.
  - Parámetros de entrada: Identificador del `rol`
  - Parámetros de salida: 0 si ha ido todo bien o -1 en caso contrario.
- `modificarRol`
  - Descripción: esta operación modifica los datos del `rol` indicado.
  - Parámetros de entrada: Datos del `rol`.

  - Parámetros de salida: 0 si ha ido todo bien o -1 en caso contrario.

- `asignarUsuarioARol`
  - Descripción: esta operación asigna un usuario a un rol.
  - Parámetros de entrada: Identificador del `rol` y del `usuario`.
  - Parámetros de salida: 0 si ha ido todo bien o -1 en caso contrario.

- `desasignarUsuarioDeRol`
  - Descripción: esta operación elimina la asignación de un usuario de un rol.
  - Parámetros de entrada: Identificador del `rol` y del `usuario`.
  - Parámetros de salida: 0 si ha ido todo bien o -1 en caso contrario.

- `buscarUsuariosDeRol`
  - Descripción: esta operación devuelve el conjunto de usuarios asignados a un rol.
  - Parámetros de entrada: Identificador del `rol`
  - Parámetros de salida: Conjunto de `usuarios` asignados a un `rol`.

- `buscarRolesDeUsuario`
  - Descripción: esta operación devuelve el conjunto de roles asignados a un usuario.
  - Parámetros de entrada: Identificador del `usuario`.
  - Parámetros de salida: Conjunto de `roles` asignados a un `usuario`.

Administración de Permisos

La especificación de las operaciones encargadas de la administración de permisos es:

- `asignarPermisosARol`
  - Descripción: esta operación asigna a un rol el permiso sobre un objeto.
  - Parámetros de entrada: Identificador del `rol` e identificador del `objeto`
  - Parámetros de salida: 0 si ha ido todo bien o -1 en caso contrario.
- `desasignarPermisosDeRol`
  - Descripción: esta operación elimina permisos a cada uno de los roles para ejecutar las operaciones que se consideren necesarias.
  - Parámetros de entrada: Identificador del `rol` e identificador del `objeto`

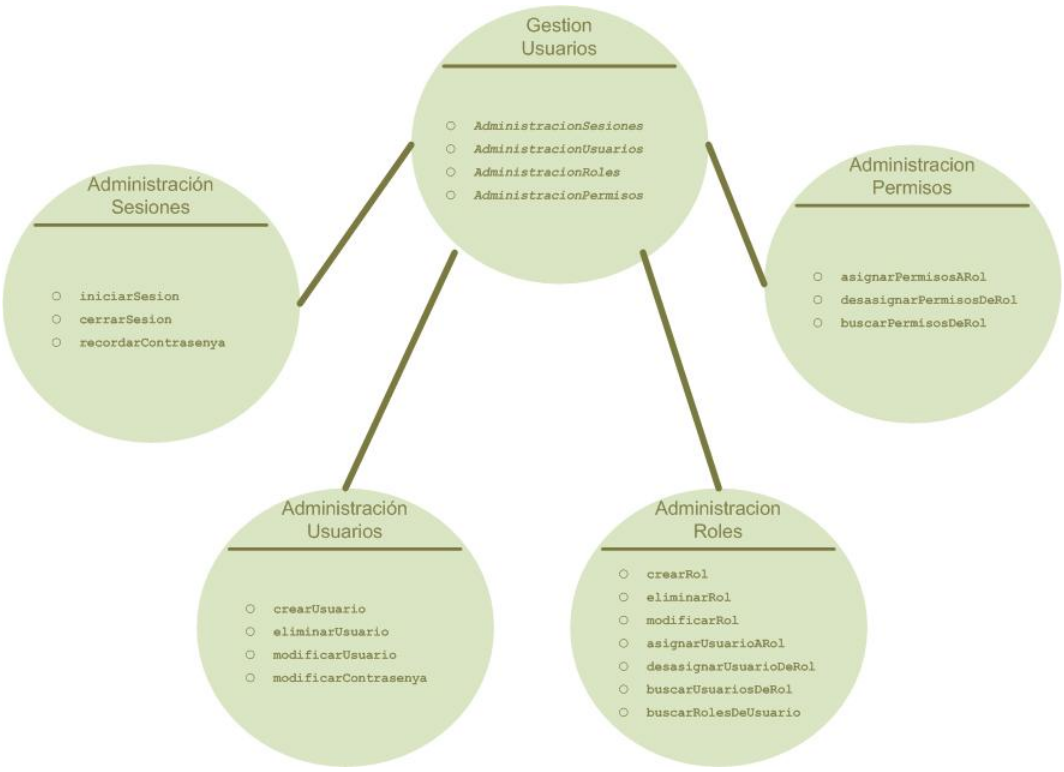

Figura 5.12: Operaciones publicadas en la *Gestión de Usuarios*

  - Parámetros de salida: 0 si ha ido todo bien o -1 en caso contrario.

- `buscarPermisosDeRol`
  - Descripción: esta operación devuelve el conjunto de permisos sobre objetos de un rol.
  - Parámetros de entrada: Identificador del `rol`
  - Parámetros de salida: Conjunto de `objetos` sobre los que tiene permiso un `rol`.

La figura 5.12 muestra un resumen de las operaciones publicadas para el grupo funcional *Gestión de Usuarios*.

5.4 Recuperación de Información

El grupo funcional *Recuperación de Información* está formado por operaciones que implementan la recuperación y búsqueda de información. Estas operaciones recuperan la información de las entidades que forman parte del sistema.

Existen dos formas de diseñar las operaciones de este grupo funcional:

- *Operaciones dependientes de la aplicación a desarrollar*. Son operaciones de grano fino que se diseñan basándose en los distintos elementos que conforman los contextos navegacionales en cada dominio.
- *Operaciones independientes de la aplicación a desarrollar*. Son operaciones de grano grueso que se diseñan independientemente de los elementos que conforman los contextos navegacionales en cada dominio. Las operaciones son genéricas y se basan únicamente en la existencia de los elementos.

Si las operaciones se diseñan dependientes de la aplicación a desarrollar, su implementación es mucho más sencilla y se puede reutilizar más fácilmente. En cambio, si el diseño es independiente de la aplicación a desarrollar, su implementación se hace más compleja y también más difícil de reutilizar.

En esta tesis es importante poder reutilizar las operaciones (se hará uso de ellas en el grupo funcional de *Soporte a la Presentación*, en la página 131). Además, para un mejor entendimiento de la propuesta de la tesis, se va a presentar el diseño de operaciones dependientes de la aplicación a desarrollar. Como trabajo futuro queda pues, la incorporación de las operaciones independientes de la aplicación a desarrollar en la herramienta `DISWOOM`.

El trabajo desarrollado en la tesis propone una serie de reglas que permiten identificar las operaciones a partir de los *contextos navegacionales*, los cuales forman parte del modelo de *Navegación* (ver apartado 4.3.2 en la página 68). Los contextos nos permiten obtener funcionalidad necesaria para nuestro servicio Web, como es la recuperación de información o la navegación permitida a un usuario. Un contexto define una vista sobre el modelo de clases, las Unidades Abstractas de Información (UAI). Permite identificar, para cada vista distinta, una operación de recuperación de información. Además, para cada contexto también se pueden definir filtros de búsqueda, índices y condiciones de población.

A continuación se describe con detalle cómo se obtienen las operaciones a partir de estos modelos.

5.4.1 Reglas de transformación

En este apartado se presentan las cuatro reglas que constituyen el diseño de las operaciones de *Recuperación de Información*. Cada regla está basada en un elemento del *contexto navegacional*. Para finalizar el apartado, se muestran los algoritmos que implementan estas reglas.

La primera regla utiliza las UAIs que conforman cada *contexto navegacional*.

REGLA RNUAI: Identificación de `Recuperar`*`NombreUAI`*

Entrada: Mapa navegacional

Por cada UAI definida en un contexto navegacional, se identifica una ope-

ración `recuperar`*`NombreUAI`*.

Salida: Lista de operaciones del tipo `recuperar`*`NombreUAI`*

En el ejemplo del caso de estudio del comercio electrónico, el contexto navegacional *Producto* está formado por dos UAI llamadas *Producto* y *Productos Similares*. De este contexto y siguiendo esta regla, se seleccionan las operaciones `recuperarProducto` y `recuperarProductosSimilares`.

Para la segunda regla se tienen en cuenta los índices que pueden definirse en los *contextos navegacionales*.

REGLA INI: Identificación de `Indexar`*`NombreIndice`*

Entrada: Mapa navegacional

Por cada índice definido en un contexto navegacional, se identifica una operación `indexar`*`NombreIndice`*.

Salida: Lista de operaciones del tipo `indexar`*`NombreIndice`*

Para el ejemplo del caso de estudio, en el contexto navegacional *Producto* está definido el índice *Producto Index*. De este contexto y siguiendo esta regla, se selecciona la operación `indexarProductoIndex`.

En la tercera regla se utilizan las condiciones de población que pueden estar definidas en cada *contexto navegacional*. Este mecanismo es el menos utilizado de los definidos en los modelos de OO-Method OOWS.

REGLA RPNC: Identificación`RecuperarPoblación`*`NombreCondicion`*

Entrada: Mapa navegacional

Por cada mecanismo de condición de población definido en un contexto

navegacional, se identifica una operación `recuperarPoblacion`*`NombreCondicion`*.

Salida: Lista de operaciones del tipo `recuperarPoblacion`*`NombreCondicion`*

Para el ejemplo del caso de estudio, no existe ninguna condición de población definida.

Para finalizar, la cuarta regla utiliza los filtros de búsqueda que pueden definirse en los *contextos navegacionales*.

REGLA BNF: Identificación de `Buscar`*`NombreFiltro`*

Entrada: Mapa navegacional

Por cada filtro de búsqueda definido en un contexto navegacional se identifica una operación `buscar`*`NombreFiltro`*.

Salida: Lista de operaciones del tipo `buscar`*`NombreFiltro`*

Para el ejemplo del caso de estudio, en el contexto navegacional *Producto* está definido el filtro de búsqueda *Producto*. De este contexto y siguiendo esta regla, se selecciona la operación `buscarProducto`.

Implementación de las reglas

La estructura de los algoritmos que se presentan a continuación está formada por: una *entrada* que indica qué espera el algoritmo que se le pase como entrada, el modelo del cuál se identifican las operaciones; y finalmente la *nomenclatura*, que son los elementos del modelo de entrada que se van a utilizar durante el algoritmo.

- Entrada:
  - Contexto Navegacional (`CN`) no vacío
- Nomenclatura:
  - `CN.NavigationalModel.AIU, iau`: UAI del contexto
  - `iau.ManagerClass`: clase principal de la UAI
  - `iau.ComplementaryClass, cc`: clase complementaria de la UAI
  - `iau.Index, indice`: índice de la UAI
  - `iau.Filter, filtro`: filtro del contexto
  - `iau.ManagerClass.PopulationCondition`: condición de población de la UAI

El algoritmo 4 implementa la *regla RNUAI* que se corresponde con la identificación de las operaciones de tipo `recuperar`*`NombreUAI`*. Para cada UIA se identifica: (1) el nombre de la UIA (que forma parte del nombre de la operación); (2) la clase principal y sus parámetros; (3) las clases complementarias y sus parámetros.

Algoritmo 4 Recuperar Nombre UAI

```
self.NavigationalModel.AIU->forEach(iau:oows.AIU) {
  //RecuperarNombreUAI
  ls_NombreUAI = iau.id
  ls_ClasePpal = iau.ManagerClass.Class.id
  iau.ManagerClass.NavigationalAttribute->forEach
    (na:oows.NavigationalAttribute) {
    if (ls_ParametrosPpal.size() > 0) {
      ls_ParametrosPpal = ls_ParametrosPpal + ',' + na.Attribute.id
    } else { ls_ParametrosPpal = na.Attribute.id }  }
  iau.ComplementaryClass -> forEach(cc:oows.ComplementaryClass) {
    ll_ClaseSec.add(cc.Class.id)
    cc.NavigationalAttribute -> forEach(cna:oows.NavigationalAttribute) {
       if (ls_ParamSec.size() > 0) {
         ls_ParamSec = ls_ParamSec + ',' + cna.Attribute.id
         } else { ls_ParamSec = cna.Attribute.id } }
    ll_ParametrosSec.add(ls_ParamSec) }
```

El algoritmo 5 implementa la *regla INI* que se corresponde con la identificación de las operaciones de tipo `indexar`*`NombreIndice`*. Para ello, se recorren las UAIs del modelo navegacional. Para cada UIA, y si esta tiene un índice definido, se identifica: (1) el nombre del índice (que forma parte del nombre de la operación); (2) la clase principal y sus parámetros.

Algoritmo 5 Identificación de `Indexar`*`NombreIndice`*

```
//IndexarNombreIndice
iau.Index -> forEach(indice:oows.Index) {
  ls_NombreIndice = indice.id
  ls_ClasePpal = indice.SearchView.AIU.ManagerClass.Class.id
  indice.Attribute -> forEach(a:oows.Attribute) {
    if (ls_ParametrosPpal.size() > 0) {
      ls_ParametrosPpal = ls_ParametrosPpal + ',' + a.id
      } else { ls_ParametrosPpal = a.id } }
```

El algoritmo 6 implementa la *regla BNF* que se corresponde con la identificación de las operaciones de tipo `buscar`*`NombreFiltro`*. Para ello, se recorren las UAIs del modelo navegacional. Para cada UIA, y si esta tiene un filtro definido, se identifica: (1) el nombre del filtro (que forma parte del nombre de la operación); (2) la clase principal y sus parámetros.

Algoritmo 6 Identificación de `buscar`*`NombreFiltro`*

```
//BuscarNombreFiltro
iau.Filter -> forEach(filtro:oows.Filter) {
  ls_NombreFiltro =  filtro.id.firstToLower()
  ls_ClasePpal = filtro.SearchView.AIU.ManagerClass.Class.id
  filtro.SearchView.SearchViewAttribute -> forEach
      (af:oows.NavigationalAttribute) {
    if(ls_ParametrosPpal.size() > 0){
      ls_ParametrosPpal = ls_ParametrosPpal + ',' + af.Attribute.id
    } else { ls_ParametrosPpal = af.Attribute.id } }
```

El algoritmo 7 implementa la *regla RPNC* que se corresponde con la identificación de las operaciones de tipo `recuperarPoblacion`*`NombreCondicion`*. Para ello, se recorren las UAIs del modelo navegacional. Para cada UIA, y si esta tiene una condición definida, se identifica: (1) el nombre de la condición

de población (que forma parte del nombre de la operación); (2) la clase principal y los parámetros de la condición; (3) las clases complementarias y sus parámetros.

Algoritmo 7 Identificación de `recuperarPoblacion`*NombreCondicion*

```
//recuperarPoblacion
if ( iau.ManagerClass.PopulationCondition.size()>0 ){
  ls_NombreCondicion = iau.ManagerClass.id
  ls_ParametrosPpal = iau.ManagerClass.PopulationCondition
  iau.ComplementaryClass -> forEach(cc:oows.ComplementaryClass){
    ll_ClaseSec.add(cc.Class.id)
    cc.NavigationalAttribute -> forEach(cna:oows.NavigationalAttribute){
      if (ls_ParamSec.size() > 0) {
        ls_ParamSec = ls_ParamSec + ',' + cna.Attribute.id
        } else { ls_ParamSec = cna.Attribute.id } }
      ll_ParametrosSec.add(ls_ParamSec) }
```

5.4.2 Ejemplo del caso de estudio

En la figura 5.13 se muestran las operaciones identificadas siguiendo los algoritmos 4, 5, 6 y 7 en el contexto *Producto*.

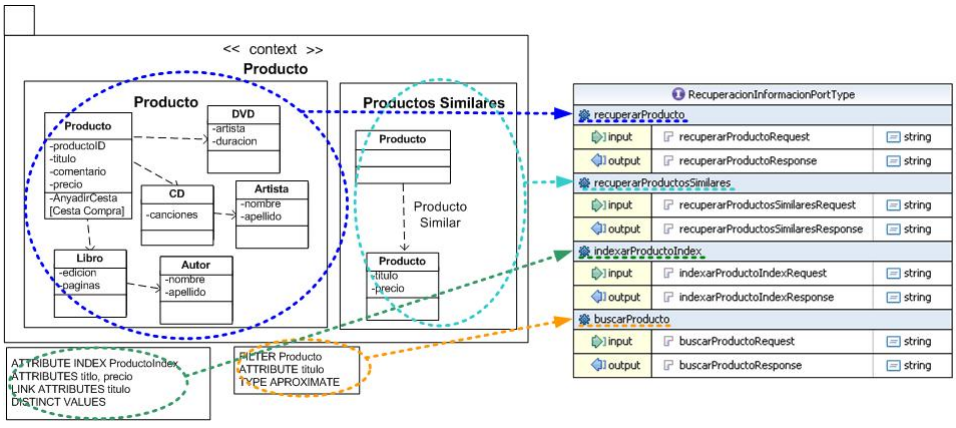

Figura 5.13: Identificación de operaciones para la *Recuperación de Información*

Siguiendo estos cuatro algoritmos en el caso de estudio, se identifican las siguientes operaciones:

- `recuperarProducto`
  - Descripción: esta operación permite recuperar la información de todos los productos (si no se le pasa ningún parámetro) o de un producto en particular (si se pasa su identificador como parámetro).
  - Parámetros de entrada: Opcionalmente el identificador del producto (`[IDProducto]`).
  - Parámetros de salida: Conjunto de productos o los datos del producto solicitado.
- `recuperarProductosSimilares`
  - Descripción: esta operación permite recuperar la información de todos los productos similares a otro producto o conjunto de productos.
  - Parámetros de entrada: Opcionalmente el identificador del producto `[IDProducto]`.
  - Parámetros de salida: Conjunto de productos.
- `recuperarCategoriasProductos`
  - Descripción: esta operación permite recuperar las categorías de los productos existentes o la categoría de un producto en particular.
  - Parámetros de entrada: Opcionalmente el identificador del producto (`[IDProducto]`).
  - Parámetros de salida: conjunto de categorías de productos o la categoría del producto solicitado.
- `recuperarComentariosClientes`
  - Descripción: esta operación permite recuperar los comentarios de los clientes sobre ciertos productos o sobre un producto en particular.
  - Parámetros de entrada: Opcionalmente el identificador del producto `[IDProducto]`.
  - Parámetros de salida: conjunto de comentarios o los comentarios del producto solicitado.

- `recuperarClientes`
  - Descripción: esta operación permite recuperar los datos de todos los clientes o de un cliente en particular.
  - Parámetros de entrada: Opcionalmente el identificador del cliente `[IDClientes]`.
  - Parámetros de salida: conjunto de clientes o los datos del cliente solicitado.
- `recuperarVendedores`
  - Descripción: esta operación permite recuperar los datos de todos los vendedores o de un vendedor en particular.
  - Parámetros de entrada: Opcionalmente el identificador del vendedor `[IDVendedores]`.
  - Parámetros de salida: conjunto de vendedores o los datos del vendedor solicitado.
- `indexarProductoIndex`
  - Descripción: esta operación provee un acceso indexado a la población de productos por su nombre y precio.
  - Parámetros de entrada: Ninguno.
  - Parámetros de salida: colección de productos indexados por el nombre y el precio.
- `buscarProducto`
  - Descripción: esta operación filtra el espacio de productos que recupera por nombre.
  - Parámetros de entrada: Nombre del producto.
  - Parámetros de salida: colección de productos que cumplen los requisitos de la búsqueda.
- `buscarVendedores`
  - Descripción: esta operación filtra el espacio de vendedores que recupera por nombre.

- Parámetros de entrada: Nombre del vendedor.
- Parámetros de salida: colección de listas de vendedores que cumplen los requisitos de la búsqueda.

Figura 5.14: Operaciones publicadas de la *Recuperación de Información*

En la figura 5.14 se muestran las operaciones publicadas para el grupo funcional *Recuperación de Información*.

5.5 Soporte a la Navegación

Este grupo funcional está formado por operaciones que implementan la navegación permitida a cada uno de los usuarios de la aplicación. Así mismo, este grupo complementa los servicios de los demás grupos funcionales que no tienen conocimiento alguno sobre la navegación permitida.

El *Soporte a la Navegación* es adecuado para aquellos desarrolladores que decidan construir clientes Web que sigan los requisitos navegacionales definidos en el modelo navegacional. Este servicio traslada la lógica de navegación al nivel de interacción, facilitando la implementación de mecanismos de personalización y adaptación de aplicaciones Web [108, 109, 110].

Las operaciones de este grupo no dependen de la aplicación a desarrollar, por lo que se puede generalizar la forma de identificarlas. El trabajo desarrollado en la tesis propone una serie de reglas que permiten identificar las

operaciones a partir de los *mapas navegacionales* y de los *contextos navegacionales*, los cuales forman parte del modelo de *Navegación* (ver apartado 4.3.2 en la página 68).

El modelo de navegación permite añadir funcionalidad necesaria a un servicio Web, como puede ser la recuperación de información o la navegación permitida a un usuario. Los mapas navegacionales tienen dos tipos de enlaces:

- *Enlaces de exploración* (representado por una flecha discontinua) que define una relación de alcanzabilidad entre cualquier contexto del mapa navegacional y el contexto donde termina el enlace.
- *Enlaces de secuencia* (representado por una flecha continua) que definen una relación de alcanzabilidad entre dos contextos (página Web en la implementación).

Por otro lado, en los contextos navegacionales se puede encontrar los *Enlaces de Servicio*. Este tipo de enlaces permiten definir el contexto navegacional al cual accede el usuario después de la ejecución de una operación.

A continuación, se describe con detalle cómo se obtienen las operaciones a partir de estos modelos.

5.5.1 Reglas de transformación

En este apartado se presentan las tres reglas que constituyen el diseño del grupo funcional *Soporte a la Navegación*. Cada regla está basada en cada uno de los tipos de enlaces que existen en los mapas navegacionales.

Los mapas navegacionales tienen forma de árbol, al igual que los modelos utilizados en la identificación de las operaciones de los grupos funcionales de *Lógica de la Aplicación* y de *Gestión de Identidad*. Por lo tanto, se van a recorrer sus nodos en preorden.

Mientras se recorren los mapas navegacionales, se busca para que cumplan las siguientes reglas.

La primera regla tiene en cuenta los enlaces de tipo exploración existentes entre los contextos navegacionales.

REGLA REE: Identificación de `RecuperarEnlaceExploracion`

Entrada: Mapa navegacional

Si existe algún enlace de exploración, entonces se identifica la operación `recuperarEnlacesExploracion`.

Salida: La operación `recuperarEnlaceExploracion`

En el ejemplo del caso de estudio del comercio electrónico, siguiendo esta regla, se encuentran 6 enlaces de exploración. Por lo tanto se identifica la operación `recuperarEnlacesExploracion`.

La segunda regla tiene en cuenta los enlaces de tipo secuencia existentes entre dos contextos navegacionales.

REGLA RES: Identificación de `RecuperarEnlaceSecuencia`

Entrada: Mapa navegacional

Si existe algún enlace de secuencia, entonces se identifica la operación `recuperarEnlacesSecuencia`.

Salida: La operación `recuperarEnlaceSecuencia`

Para el ejemplo del caso de estudio, existen 4 enlaces de secuencia, por lo tanto se identifica la operación `recuperarEnlacesSecuencia`.

Por último, se buscan los enlaces de servicios que se encuentran definidos dentro de los contextos navegacionales.

REGLA RESV: Identificación de `RecuperarEnlaceServicio`

Entrada: Mapa navegacional

Se recorren los contextos navegacionales, y si existe algún enlace de servicio, entonces se identifica la operación `recuperarEnlacesServicio`.

Salida: La operación `recuperarEnlaceServicio`

Siguiendo con el caso de estudio, en el contexto *Producto* está definido un enlace de servicio para el método `anyadirCesta`. Este enlace dirige la navegación al contexto *Cesta Compra*, tras la ejecución del método. Por lo tanto, se identifica la operación `recuperarEnlacesServicio`.

Implementación de las reglas

Los algoritmos que se presentan a continuación están formados por: una *entrada* que indica qué espera el algoritmo que se le pase como entrada, el modelo del cuál se identifican las operaciones; la *nomenclatura*, que son elementos del modelo de entrada que se van a utilizar durante el algoritmo; y finalmente las *funciones*, que son operaciones que realizan una acción en concreto y ayudan a simplificar el funcionamiento del algoritmo.

- Entrada:
  - Mapa Navegacional (`MN`)
- Nomenclatura:
  - `MN.UserRol, ur`: Rol de usuario del navegacional del mapa
  - `ur.NavigationalNode`: contexto navegacional
- Funciones:

– `navegacionContextos`: devuelve la colección `coleccionEnlaceExploracion` con los enlaces de exploración y la colección `coleccionEnlaceSecuencia` con los enlaces de secuencia accesibles desde el *Contexto Navegacional* del cual se llama.

El algoritmo 8 implementa la *regla REE*, que se corresponde con la identificación de las operaciones de tipo `recuperarEnlaceExploracion`. Para ello, se recorren los contextos navegacionales en busca de enlaces de tipo exploración. Si existe alguno, se publica la operación.

Algoritmo 8 Recuperar Enlace Exploración

```
self.NavigationalModel.UserRol -> forEach(ur:oows.UserRol) {
    ur.NavigationalNode -> forEach(nn:oows.NavigationalContext) {
        nn.navegacionContextos() }
    if (not coleccionEnlaceExploracion.isEmpty()) {
        <%  public String recuperarEnlacesExploracion() %> } }
```

El algoritmo 9 implementa la *regla RES*, que se corresponde con la identificación de las operaciones de tipo `recuperarEnlaceSecuencia`. Para ello, se recorren los contextos navegacionales en busca de enlaces de tipo secuencia. Si existe alguno, se publica la operación.

Algoritmo 9 Recuperar Enlace Secuencia

```
self.NavigationalModel.UserRol -> forEach(ur:oows.UserRol) {
    ur.NavigationalNode->forEach(nn:oows.NavigationalSubsystem) {
     nn.NavigationalNode->forEach(nnnn:oows.NavigationalContext){
       nnnn.navegacionContextos() } }
    if (not coleccionEnlaceSecuencia.isEmpty()){
        <%public String recuperarEnlaceSecuencia
        °(String ps_servicio)} %> } }
```

El algoritmo 10 implementa la *regla RESV*, que se corresponde con la identificación de las operaciones de tipo `recuperarEnlaceServicio`. Para ello, se recorren los contextos navegacionales en busca de operaciones navegacionales. Si existe alguna, se publica la operación.

Algoritmo 10 Recuperar Enlace Servicio

```
self.NavigationalModel.UserRol -> forEach(ur:oows.UserRol) {
    ur.NavigationalNode->forEach(no:oows.NavigationalOperation) {
        <% public String recuperarEnlaceServicio
            (String ps_servicio){ %> } }
```

5.5.2 Ejemplo del caso de estudio

La especificación de las operaciones encargadas del soporte a la navegación, para el caso de estudio se presenta a continuación. En la figura 5.15 se muestran las operaciones y desde qué elementos de los contextos y de los mapas navegacionales han sido identificadas siguiendo los algoritmos 8, 9 y 10.

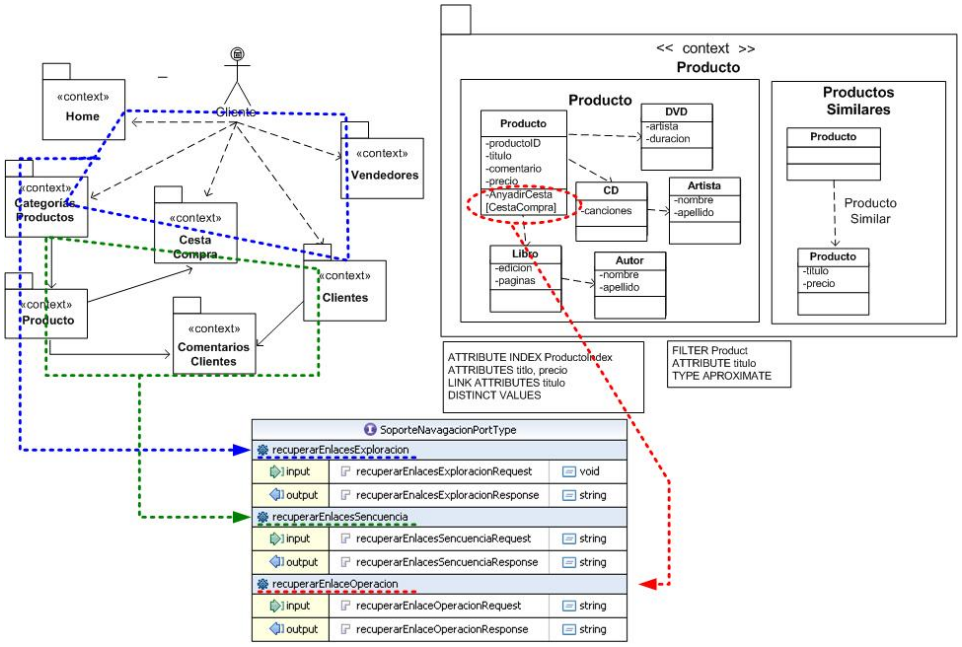

Figura 5.15: Identificación de operaciones para el *Soporte a la Navegación*

- `recuperarEnlacesExploracion`
  - Descripción: esta operación permite recuperar los enlaces de exploración alcanzables por el usuario. Estos enlaces se pueden activar

desde cualquier contexto del mapa navegacional.

- Parámetros de entrada: Ninguno.
- Parámetros de salida: Conjunto de contextos alcanzables por el usuario en la aplicación a través de un enlace de exploración.

- `recuperarEnlacesSecuencia`
  - Descripción: esta operación permite recuperar los enlaces de secuencia alcanzables desde un contexto concreto por el usuario.
  - Parámetros de entrada: Nombre del contexto.
  - Parámetros de salida: Conjunto de contextos alcanzables por el usuario desde el contexto indicado.

- `recuperarEnlaceServicio`
  - Descripción: esta operación permite recuperar el enlace alcanzable después de ejecutar una operación.
  - Parámetros de entrada: Nombre del servicio.
  - Parámetros de salida: Enlace al contexto alcanzable tras la ejecución del servicio.

En la figura 5.16 se muestran las operaciones publicadas para el grupo funcional *Soporte a la Navegación*.

Figura 5.16: Operaciones publicadas del *Soporte a la Navegación*

5.6 Soporte a la Presentación

Este grupo funcional está formado por dos operaciones: la primera de ellas permite recuperar datos con unas propiedades de presentación específicas, la segunda permite mostrar al usuario una página Web completa. Este grupo incorpora a los servicios Web la funcionalidad que proveen los portlets [102] o mashups [111, 112, 113, 114, 103, 115].

Las operaciones de este grupo, al igual que las operaciones del grupo *Recuperación de Información* (en la página 115), pueden ser operaciones dependientes de la aplicación a desarrollar u operaciones independientes de la aplicación a desarrollar.

Como se ha comentado en el apartado 5.4, en esta tesis se van a utilizar operaciones dependientes de la aplicación a desarrollar puesto que permiten que se entienda mejor la propuesta, son reutilizables y más fáciles de implementar. En trabajos futuros, se abordará la incorporación de las operaciones independientes de la aplicación en la herramienta `DISWOOM`.

El trabajo desarrollado en la tesis propone una serie de reglas que permiten identificar las operaciones a partir de dos modelos:

- Por una parte, las propiedades de presentación utilizadas se basan en los modelos de Presentación (ver apartado 4.3.2 en la página 71).
- Por otro lado, las operaciones de este grupo se identifican utilizando los *contextos navegacionales*, los cuales forman parte del modelo de *Navegación* (ver apartado 4.3.2 en la página 68).

A continuación, se describe con detalle cómo se obtienen las operaciones a partir de estos modelos.

5.6.1 Reglas de transformación

En este apartado se presenta las dos reglas que constituyen el diseño del grupo funcional *Soporte a la Presentación*. Ambas reglas están basadas en los modelos

de presentación, pero además, la primera regla utiliza las operaciones identificadas en los anteriores grupos funcionales. Por su parte, la segunda regla utiliza los mapas navegacionales.

La primera regla es una extensión que permite incorporar propiedades de presentación a los datos que se obtienen por medio de las operaciones de los grupos presentados anteriormente: *Lógica de la Aplicación* en la página 89, *Gestión de Usuarios* en la página 102 y *Recuperación de Información* en la página 115.

REGLA PIN: Identificación de `PresentacionDeInformacion`

Entrada: Ninguna

Si se ha definido alguna operación para el servicio Web (en cualquiera de los grupos funcionales de *Lógica de la Aplicación*, *Gestión de Usuarios*, *Recuperación de Información*) o *Soporte a la Navegación*, entonces se identifica la operación `presentacionDeInformacion`.

Salida: La operación `presentacionDeInformacion`

Por ejemplo, la operación `buscarProducto` devuelve la información definida en la búsqueda del contexto *Producto* (ver la definición de la operación en apartado 5.4.2). Si se llama directamente a esta operación, se obtienen los productos en un documento XML que se muestra en un navegador Web como se puede ver en la figura 5.17.

```
- <Productos diffgr:id="Productos1" msdata:rowOrder="0">
    <_oid>6</_oid>
    <nombre>Buffy la Cazavampiros: La colección completa en DVD</nombre>
    <autor>Sarah Michelle Gellar, Antony Stewart Head and Nicholas Brendon</autor>
    <precio>70</precio>
  </Productos>
- <Productos diffgr:id="Productos2" msdata:rowOrder="1">
    <_oid>25</_oid>
    <nombre>Buffy la Cazavampiros: El álbum</nombre>
    <autor>Original Soundtrack</autor>
    <precio>15</precio>
  </Productos>
```

Figura 5.17: Documento XML devuelto por `buscarProducto`

En cambio, si se llama a esta operación a través de `presentacionDeInformacion` se obtienen los mismos productos pero con propiedades de presentación y se muestra en un navegador Web como se puede ver en la figura 5.18. La lista de productos devuelta por ambas operaciones es la misma, sólo que en la segunda lista (la devuelta por la operación `presentacionDeInformacion`) se han asociado las propiedades de presentación definidas en el modelo de presentación.

Figura 5.18: Resultado de `presentacionDeInformacion(buscarProducto)`

Como se ha comentado, es posible que los usuarios del servicio Web indiquen parámetros de presentación para aplicarlos en la operación `presentacionDeInformacion` indicando los patrones de presentación a ser aplicados. Si estos parámetros se indican, entonces las propiedades del modelo de presentación no se tienen en cuenta. Esta posibilidad facilita a los consumidores de servicios Web que quieran adaptar las propiedades de presentación a sus preferencias.

La segunda de las extensiones que se propone, es la definición de una operación por *contexto navegacional*, que implementa completamente la página Web equivalente al contexto con la presentación definida durante la etapa de modelado. Esta operación hace uso de las operaciones de los otros grupos funcionales para poder devolver la página Web completa:

- `recuperar`*`NombreUAI`* para cada una UAI definida en el contexto.
- `recuperarEnlacesExploracion` para recuperar los enlaces a otras páginas.

Para identificar las operaciones, se recorre el modelo de Navegación siguiendo la siguiente regla:

REGLA PNC: Identificación de `PresentacionNombreContexto`

Entrada: Mapa navegacional

Por cada contexto navegacional se identifica una operación `presentacionDe`*NombreContexto*.

Salida: Lista de operaciones del tipo `presentacion`*NombreContexto*

En el ejemplo del caso de estudio del comercio electrónico, se encuentra el contexto navegacional *Producto* por lo que se define la operación `presentacionDeProducto`. Esta operación devuelve una página Web equivalente a las propiedades definidas por el modelador en dicho contexto (en la figura 5.19 se puede ver un ejemplo del resultado devuelto por esta operación).

Figura 5.19: Resultado de `presentacionDeProducto`

Implementación de las reglas

El algoritmo que se presenta a continuación está formado por: una *entrada* que indica qué espera el algoritmo que se le pase como entrada, el modelo del cuál se identifican las operaciones; y finalmente la *nomenclatura*, que son elementos del modelo de entrada que se van a utilizar durante el algoritmo.

- Entrada:
  - Mapa Navegacional (`MN`)
- Nomenclatura:
  - `MN.UserRol, ur`: Rol de usuario del navegacional del mapa
  - `ur.NavigationalNode`: contexto navegacional

El algoritmo 11 implementa la *regla PIN* y la *regla PNC*. Para la primera regla, se publica la operación para su utilización. Para la segunda regla se recorren el modelo navegacional de cada usuario y cada uno de los contextos que lo forman.

Algoritmo 11 Presentación de Información y Presentación Nombre Contexto

```
 //presentacionDeInformacion
  <% public HTMLDocument presentacionDeInformacion (String ps_operacion,
     String[] ps_parametrosOperacion, String ps_paginacion,
     String ps_orden, String ps_instancias){ %>
//presentacionDeNombreContesto
self.NavigationalModel.UserRol->forEach(ur:oows.UserRol) {
  ur.NavigationalNode->forEach(nn:oows.NavigationalNode) {
     <% public HTMLDocument presentacionDe%> nn.id <%()
```

5.6.2 Ejemplo del caso de estudio

Siguiendo el algoritmo 11 en el caso de estudio, se identifican las siguientes operaciones (ver figura 5.20):

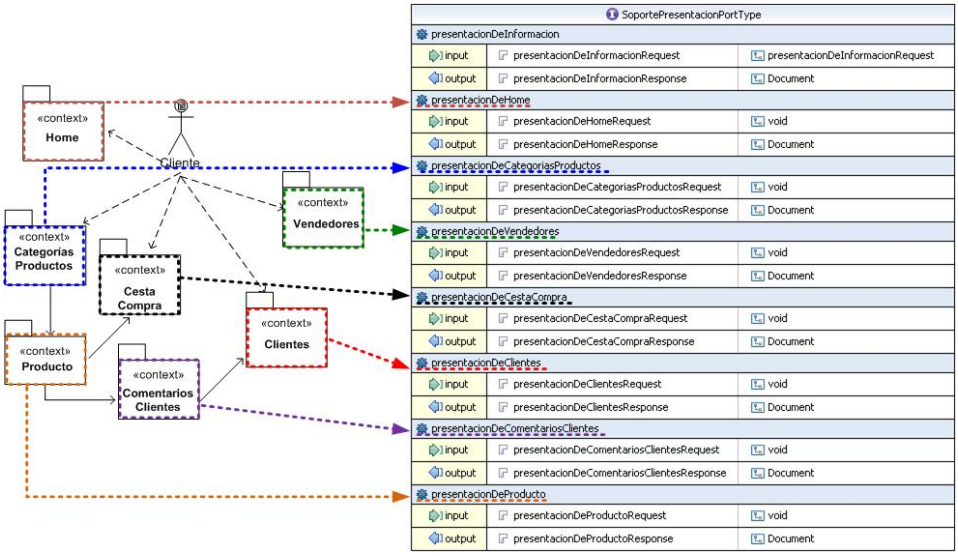

Figura 5.20: Identificación de operaciones para el *Soporte a la Presentación*

- `presentacionDeInformacion`
  - Descripción: esta operación permite incorporar propiedades de presentación a los datos que se obtiene por medio de las operaciones de los grupos presentados anteriormente.
  - Parámetros de entrada: el nombre de la operación que se desea invocar (`operacion`) junto con los parámetros de entrada que necesita (`parametrosOperacion`). Opcionalmente, se pueden indicar los patrones de presentación de información (`paginacion`, `orden` y `presentacionInstancia`) con los que se deben mostrar los datos a devolver.
  - Parámetros de salida: el conjunto de datos con presentación incluida en formato html.
- `presentacionDeHome`
  - Descripción: esta operación permite obtener una página Web que representa al contexto *Home*.
  - Parámetros de entrada: ninguno.
  - Parámetros de salida: página Web en formato html.

- `presentacionDeCategoriasProductos`
  - Descripción: esta operación permite obtener una página Web que representa al contexto *CategoriasProductos*.
  - Parámetros de entrada: ninguno.
  - Parámetros de salida: página Web en formato html.
- `presentacionDeProducto`
  - Descripción: esta operación permite obtener una página Web que representa al contexto *Producto*.
  - Parámetros de entrada: ninguno.
  - Parámetros de salida: página Web en formato html.
- `presentacionDeCestaCompra`
  - Descripción: esta operación permite obtener una página Web que representa al contexto *Cesta Compra*.
  - Parámetros de entrada: ninguno.
  - Parámetros de salida: página Web en formato html.
- `presentacionDeComentariosClientes`
  - Descripción: esta operación permite obtener una página Web que representa al contexto *ComentariosClientes*.
  - Parámetros de entrada: ninguno.
  - Parámetros de salida: página Web en formato html.
- `presentacionDeClientes`
  - Descripción: esta operación permite obtener una página Web que representa al contexto *Clientes*.
  - Parámetros de entrada: ninguno.
  - Parámetros de salida: página Web en formato html.

Figura 5.21: Operaciones publicadas del *Soporte a la Presentación*

- `presentacionDeVendedores`
  - Descripción: esta operación permite obtener una página Web que representa al contexto *Vendedores*.
  - Parámetros de entrada: ninguno.
  - Parámetros de salida: página Web en formato html.

En la figura 5.21 se muestran las operaciones publicadas para el grupo funcional *Soporte a la Presentación*.

5.7 Conclusiones

En este capítulo se ha presentado un método para la identificación automática de las operaciones de los servicios Web a partir de los modelos conceptuales de OO-Method / OOWS. Este método se ha definido partiendo de los grupos funcionales que categorizan las operaciones en base a la funcionalidad ofrecida.

El método presentado está abierto a la incorporación de nuevos grupos funcionales. Cada uno de los grupos funcionales ha sido presentado, identificándose sus operaciones, parámetros y los modelos que proporcionan información relevante para dicha identificación.

Para finalizar el capítulo, en la tabla 5.2, se puede ver un resumen de las principales características de de cada uno de los grupos funcionales.

Tabla 5.2: Principales características de los grupos funcionales

	Característica			
Grupo Funcional	Dependencia dominio	Disciplinas involucradas	Identificación operaciones	Identificación parámetros
Lógica Aplicación	Si	Elicitación requisitos	Modelo tareas	Descripción tareas
Gestión Usuarios	No	Modelado conceptual	Modelo Usuarios y RBAC	-
Recuperación Información	Si	Modelado conceptual	Contexto Navegacional	-
Soporte Navegación	No	Modelado conceptual	Mapa Navegacional	-
Soporte Presentación	Si	Modelado conceptual	Mapa Navegacional	Contexto Presentación

Capítulo 6

Implementación de los servicios Web

La propuesta metodológica OO-Method / OOWS tiene implementados sus generadores de código en arquitecturas multicapa (OlivaNOVA Model Execution [92, 91] y OOWS Suite [12] respectivamente). La representación software que propone esta tesis, implementa una capa de servicios Web sobre el estilo arquitectónico de tres capas generado para las aplicaciones OO-Method / OOWS.

En la tesis se sigue una estrategia de generación automática de código para los servicios Web (presentada en el capítulo 7), que utiliza el framework WIF [93, 12] desarrollado para OOWS Suite. En este capítulo se presenta la implementación de los servicios Web utilizando el código generado por el framework WIF y la herramienta OlivaNOVA.

Aunque la implementación de los servicios Web no es el principal objetivo de la tesis, los detalles de implementación son importantes para completar y comprender la propuesta general.

El capítulo se estructura como sigue: en primer lugar, se comentan brevemente la tecnología utilizada en la generación de código. A continuación, se introduce el framework WIF, detallando cada una de las operaciones que expone (lo que permite comprender mejor la generación de código). En tercer lugar

se presenta la estrategia de implementación basada en el framework WIF. En este mismo apartado y para cada uno de los grupos funcionales, se presenta cómo generar la implementación de las operaciones de los servicios Web. Se finaliza el capítulo con las conclusiones.

Para ejemplificar los diferentes subapartados se continúa aplicando la propuesta al ejemplo del apéndice A. En este ejemplo se modela el caso de estudio de una aplicación de comercio electrónico como *Amazon*.

6.1 Generación de Código

La propuesta presentada para la generación del código de las operaciones de los servicios Web se realiza mediante una transformación *Modelo-A-Texto*. Para la implementación de los servicios Web presentados en esta tesis, se ha optado por la tecnología Java y SOAP.

La figura 6.1 representa de forma gráfica la estrategia utilizada para la generación de código desde la perspectiva de MDA. A partir de los modelos de OO-Method / OOWS (*PIM*), se realiza una transformación (definida en función de un conjunto de correspondencias) y se obtiene el código que implementa el modelo *PIM*.

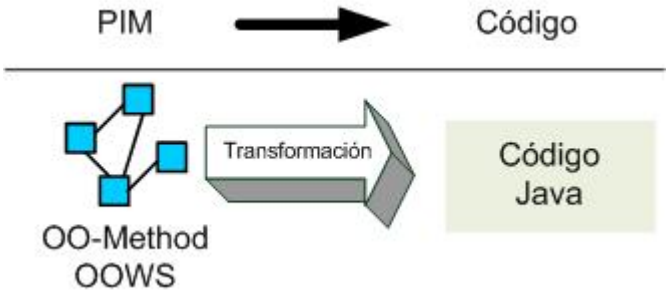

Figura 6.1: Propuesta de la generación de código desde el prisma de MDA

Los pasos seguidos para la generación de código son los siguientes:

1. Generación de la interfaz WSDL.

2. Generación del código Java.

   (a) Generación de la cabecera de la operación (identificada durante el diseño de los servicios Web en el capítulo 5).

   (b) Generación del detalle del cuerpo de la operación (para cada grupo funcional se utilizan una o varias operaciones del framework WIF). Este detalle se corresponde con la lógica de negocio que implementa las operaciones de los WSDL.

A continuación, se muestra cómo implementar el código Java de cada una de las operaciones. Para implementarlas, se ha utilizado el framework WIF desarrollado para la herramienta que da soporte a OOWS. Este framework permite utilizar, simplificando la implementación del servicio Web, las operaciones generadas por OlivaNOVA. En la figura 6.2 se muestra gráficamente cómo los servicios Web utilizan, a través del framework WIF, la funcionalidad implementada para las aplicaciones OlivaNOVA y para las aplicaciones OOWS.

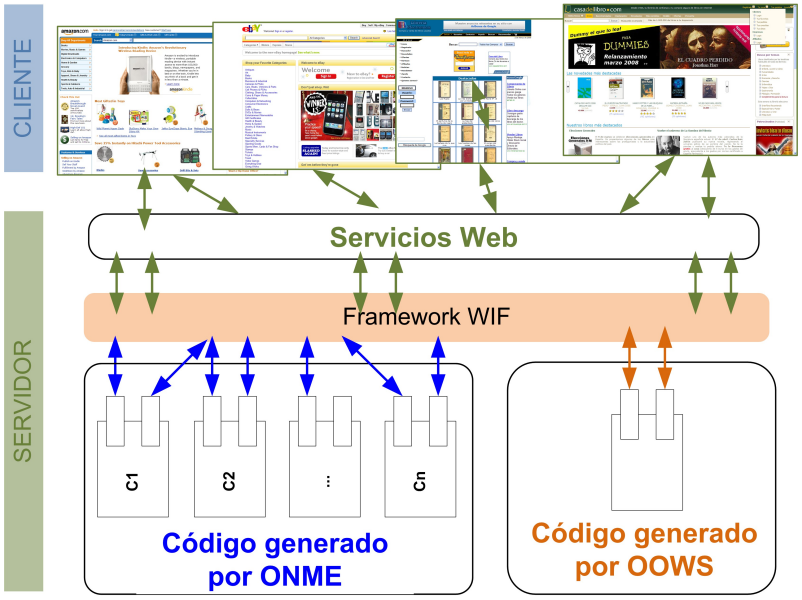

Figura 6.2: Utilización del Framework WIF

6.2 Framework WIF

La propuesta de implementación que se presenta en este capítulo, hace uso del framework WIF desarrollado para la herramienta que da soporte a OOWS [93, 12]. El framework facilita la implementación de las operaciones de los servicios Web, permitiendo su conexión e integración con la capa de aplicación generada por OlivaNOVA y con la capa de presentación generada por OOWS. Este framework es necesario dado que la utilización de la lógica de negocio generada por OlivaNOVA, resulta compleja por la gran cantidad de clases y métodos que la componen. De esta manera se consigue una implementación más natural de las operaciones y se oculta, todo lo posible, la complejidad de la lógica de negocio.

La arquitectura del framework WIF ha sido diseñada para que sea lo más independiente posible de la lógica de negocio. En esta tesis, se utiliza la lógica de negocio generada mediante OO-Method/OlivaNOVA. En la figura 6.3 se muestra cómo los servicios Web hacen uso del framework WIF.

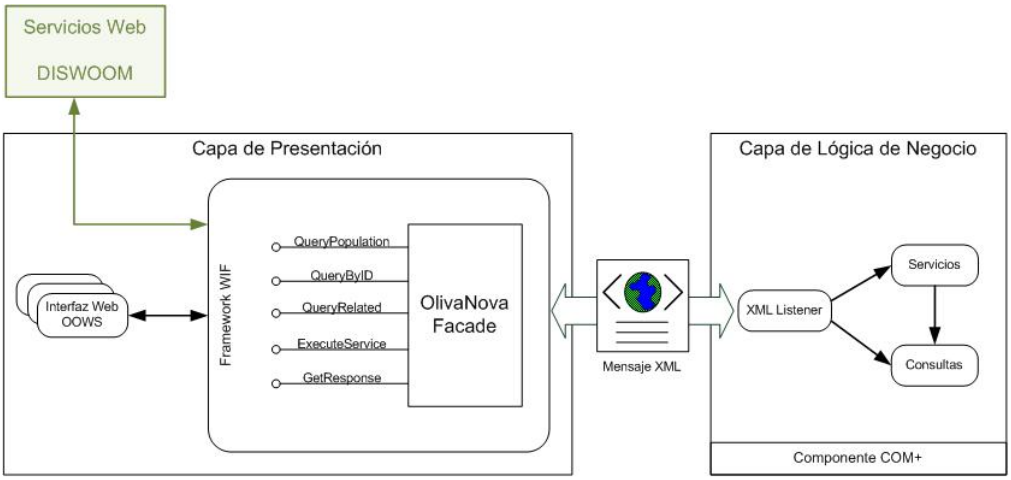

Figura 6.3: Integración de los servicios Web, el framework WIF y la lógica de negocio de OlivaNOVA

Para desacoplar la lógica de negocio, la solución adoptada por el framework WIF ha sido aplicar el patrón fachada de negocio ("facade") de Gamma [116]. El objetivo de este patrón es el de ofrecer una interfaz única de un subsistema

para facilitar su uso. En el caso del framework WIF, el subsistema está formado por el conjunto de clases que proporcionan la funcionalidad. Utilizando una única interfaz predefinida, no sólo se simplifican las interacciones, sino que para el resto de clases del framework, es transparente como se implementa el subsistema.

Para que las distintas lógicas de negocio definan la misma fachada de negocio, se ha definido en el framework la interfaz *Business Facade*, que implementa un objeto fachada concreto. Esta interfaz expone los siguientes métodos que serán los utilizados por los servicios Web:

- `QueryPopulation(class, attributes, [filter])`: devuelve la población de una clase determinada. Recibe como argumentos el identificador de la clase, una colección con los identificadores de los atributos a mostrar y un conjunto de filtros para restringir las instancias a partir de condiciones de filtrado. Un ejemplo de consulta que puede realizarse mediante este método es recuperar el título y el precio (atributos) de todos los productos (clase) del año 2008 (filtro).

- `QueryById(class, attributes, oid)`: devuelve una única instancia de una clase a partir de su identificador único `oid`. Recibe como argumentos el identificador de la clase, una colección con los identificadores de los atributos a mostrar y el identificador único de la instancia a recuperar. Un ejemplo de consulta sería recuperar el título y el precio (atributos) del producto (clase) con el identificador (oid) 115.

- `QueryRelated(class, attributes, oid, relatedClass, [filter])`: a partir del `oid` de una instancia, devuelve todas las instancias pertenecientes a otra clase relacionada. Recibe como argumentos el identificador de la clase origen, una colección con los identificadores de los atributos a mostrar de la clase relacionada, el identificador único de la instancia de la clase origen, el identificador de la relación que determina la relación entre las clases origen y destino y opcionalmente una colección de filtros. Una consulta que se realiza mediante este método es recuperar los nombres (atributo) de todos los productos similares (clase relacionada) del producto (clase) con oid 115, que se obtienen a través de la relación con la clase producto denominada *ProductosSimilares*.

- `ExecuteService(operation, arguments)`: ejecuta una operación perteneciente a la lógica de negocio a partir del identificador unívoco de un servicio y el conjunto de los argumentos. Como argumento se envía el valor (codificado como una cadena de texto). El método retorna "cierto" si la ejecución del servicio fue correcta o "falso" en caso contrario.

- `GetResponse`: recupera el resultado o el error del último método ejecutado por el framework WIF en la fachada de negocio. No recibe argumentos.

La figura 6.4 resume el proceso de integración entre OlivaNOVA, el framework WIF y los servicios Web. Se puede observar que los servicios Web de DISWOOM se integran con la lógica generada por OlivaNOVA a través del framework WIF de la herramienta OOWS Suite.

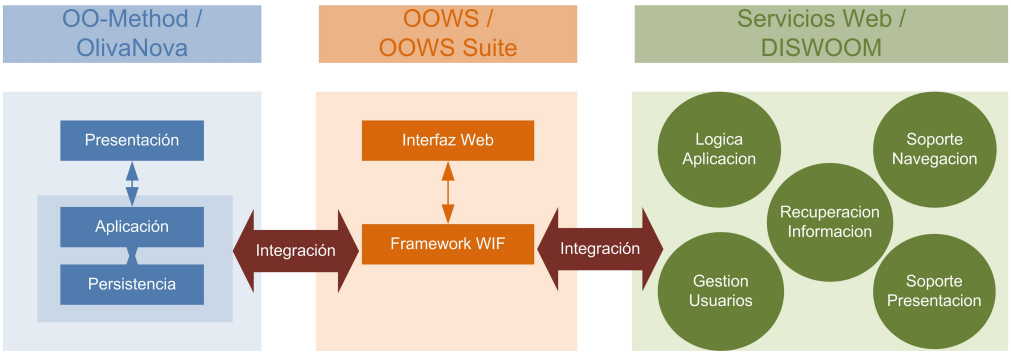

Figura 6.4: Integración entre la lógica de negocio de OlivaNOVA, el framework WIF y los grupos funcionales que forman los servicios Web

6.3 Implementación de los grupos funcionales

En este apartado se presenta la estrategia de implementación seguida en el desarrollo de la tesis. El apartado comienza con la visión general de cómo se utilizan los cinco tipos de operación del framework WIF para la implementación de los servicios Web. A continuación se detalla, para cada grupo funcional, una descripción de la estrategia particular utilizada en cada caso: primero se definen las reglas para la implementación de cada grupo, después se

presenta la implementación en MOFScript de las reglas y finalmente muestra el resultado de ejecutar dichas reglas en el caso de estudio.

Implementación utilizando el método QP: `QueryPopulation`

A continuación, se presenta cómo implementar por medio del framework WIF, aquellas operaciones del servicio Web que utilizan el método `QueryPopulation`. Se utiliza este método en aquellas operaciones que recuperan toda la población de un objeto, como por ejemplo `recuperarProducto`[1] del grupo funcional *Recuperación de Información*.

La estrategia a seguir para implementar la operación del servicio Web es la siguiente:

- Se llama al método `QueryPopulation` pasándole como parámetros el nombre de la clase, los atributos a recuperar y opcionalmente el filtro a aplicar.
- Este método devuelve directamente todas las instancias del sistema de la clase indicada.

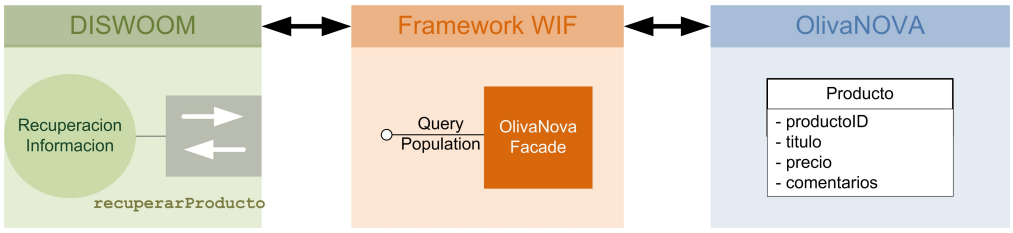

Figura 6.5: Implementación de la estrategia QP: `QueryPopulation`

En la figura 6.5, se muestra el ejemplo de la operación `recuperarProducto` cuando se solicitan todas las instancias del sistema. Para esta operación se llama al método `QueryPopulation` pasándole como parámetros:

- el nombre de la clase: `producto`;

[1]Esta operación tiene como parámetro opcional el identificador. Si no se le pasa ningún identificador, devuelve todas las instancias de `Producto`

- los atributos a recuperar: `productoID, titulo, precio, comentarios`.

Para esta estrategia existen dos posibles implementaciones, dependiendo de si se le pasa o no el filtro a aplicar:

1. En la primera implementación se le pasan el nombre de la clase y los atributos a recuperar (no se le pasa el filtro a aplicar).

```
frameworkWIF.queryPopulation(nombreClase, atributosARecuperar);
```

2. A la segunda implementación se le pasan el nombre de la clase, los atributos a recuperar y el filtro a aplicar.

```
frameworkWIF.queryPopulation(nombreClase, atributosARecuperar,
   filtro);
```

Implementación utilizando el método QBI: `QueryById`

La siguiente estrategia presenta cómo implementar, por medio del framework WIF, aquellas operaciones del servicio Web que utilizan el método `QueryById` en su implementación. Se utiliza este método en aquellas operaciones que recuperan una instancia concreta de un objeto, como por ejemplo `recuperarProducto`[2] del grupo funcional *Recuperación de Información*.

La estrategia a seguir para implementar la operación del servicio Web es la siguiente:

- Se Llama al método `QueryById` pasándole como parámetros el nombre de la clase, los atributos a recuperar y el identificador de la instancia a recuperar.

- Este método devuelve directamente la instancia indicada.

[2]Esta operación tiene como parámetro opcional el identificador. Si se le pasa un identificador, devuelve la instancia del objeto `Producto` con dicho identificador

En la figura 6.6, se muestra el ejemplo de la operación `recuperarProducto` cuando se le solicita una única instancia. Para esta operación se llama al método `QueryPopulation` pasándole como parámetros:

- el nombre de la clase: `producto`;
- los atributos a recuperar: `productoID, titulo, precio, comentarios`;
- el identificador de la instancia a recuperar.

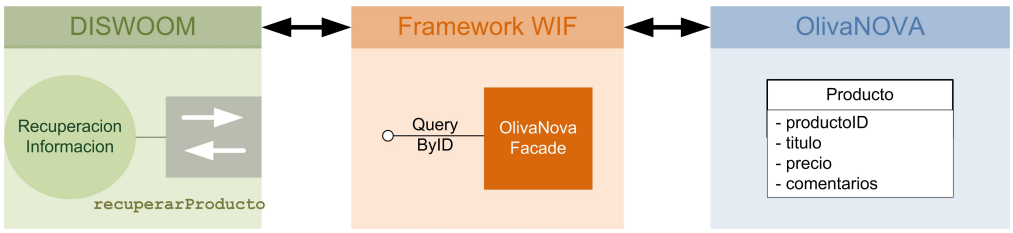

Figura 6.6: Implementación de la estrategia QBI: `QueryById`

En la implementación de esta estrategia se le pasan como parámetros el nombre de la clase, los atributos a recuperar y el identificador de la instancia que se desea recuperar.

```
frameworkWIF.queryById(nombreClase, atributosARecuperar,
    identificadorARecuperar);
```

Implementación utilizando el método QR: `QueryRelated`

La estrategia QP muestra cómo implementar, por medio del framework WIF, aquellas operaciones del servicio Web que utilizan el método `QueryRelated`. Se utiliza este método en aquellas operaciones que recuperan las instancias relacionadas con una instancia concreta de un objeto, como por ejemplo `recuperarProductosSimilares` del grupo funcional *Recuperación de Información*.

La estrategia a seguir para implementar la operación del servicio Web es la siguiente:

- Se llama al método `QueryRelated` pasándole como parámetros el nombre de la clase origen (con la que se desea relacionar las instancias), los

atributos a recuperar de la clase relacionada, el identificador de la instancia origen y el nombre de la relación con la clase relacionada que se desea recuperar.

- Este método devuelve directamente todas las instancias relacionadas con el objeto indicado.

En la figura 6.7, se muestra el ejemplo de la operación `recuperarProductosSimilares`. Para esta operación se llama al método `QueryRelated` pasándole como parámetros:

- el nombre de la clase origen: `producto`;
- los atributos a recuperar: `titulo, precio`;
- el identificador de la instancia origen;
- el nombre de la relación con la clase relacionada que se desea recuperar: `productosSimilares`

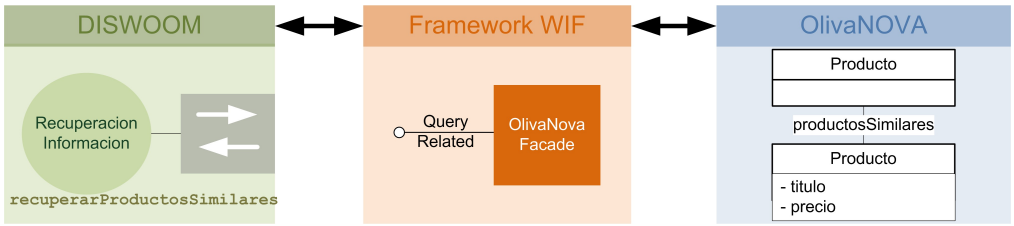

Figura 6.7: Implementación de la estrategia QP: `QueryRelated`

Para esta estrategia existen dos posibles implementaciones, dependiendo de si se le pasa o no el filtro a aplicar:

1. A la primera implementación se le pasan como parámetros el nombre de la clase, los atributos a recuperar, el identificador de la instancia de la clase y el nombre de la relación con la clase a recuperar (no se le pasa el filtro a aplicar).

```
frameworkWIF.queryRelated(nombreClase, atributosARecuperar,
   identificadorARecuperar, nombreRelacion);
```

2. En la segunda implementación se le pasan como parámetros el nombre de la clase, los atributos a recuperar, el identificador de la instancia de la clase, el nombre de la relación con la clase a recuperar y el filtro a aplicar.

```
frameworkWIF.queryRelated(nombreClase, atributosARecuperar,
   identificadorARecuperar, nombreRelacion, filtro);
```

Implementación utilizando el método ES: `ExecuteService`

La siguiente estrategia, presenta cómo implementar, por medio del framework WIF, aquellas operaciones del servicio Web que utilizan el método `ExecuteService` y que no esperan recibir una respuesta. Se utiliza este método en aquellas operaciones que llaman a una operación concreta de un objeto de OlivaNOVA, como por ejemplo `recordarContrasenya` del grupo funcional *Gestión de Usuarios*.

La estrategia a seguir para implementar la operación del servicio Web es la siguiente:

- Se llama al método `ExecuteService` pasándole como parámetros el nombre de la operación y los argumentos que necesita la operación.
- Este método retorna "cierto" si la ejecución fue correcta o "falso" en caso contrario.

En la figura 6.8, se muestra el ejemplo de la operación `recordarContrasenya`. Para esta operación se llama al método `ExecuteService` pasándole como parámetros:

- el nombre de la operación en OlivaNOVA: `MV_RemindPassword`;
- los argumentos de la operación: `nombreUsuario`.

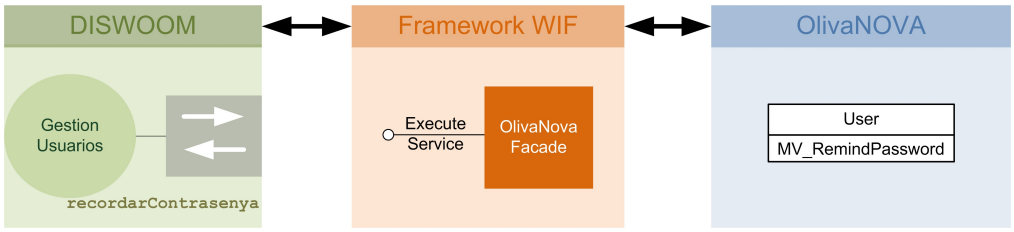

Figura 6.8: Implementación de la estrategia ES: `ExecuteService`

En la implementación de esta estrategia se pasan como parámetros el nombre de la clase, método a ejecutar y los parámetros del método.

```
frameworkWIF.executeService(nombreClase.nombreMetodo,
    parametrosMetodo);
```

Implementación utilizando los métodos ESGR: `ExecuteService` ***y*** `GetResponse`

La estrategia ESGR muestra cómo implementar, por medio del framework WIF, aquellas operaciones del servicio Web que utilizan el método `ExecuteService` y esperan recibir una respuesta. Se utiliza este método en aquellas operaciones que llaman a una operación concreta de un objeto de OlivaNOVA, como por ejemplo `modificarCestaCompra` del grupo funcional *Lógica de la Aplicación*.

La estrategia a seguir para implementar la operación del servicio Web es la siguiente:

- Se Llama al método `ExecuteService` pasándole como parámetros el nombre de la operación y los argumentos que necesita.
- Si el método retorna cierto (la ejecución del método fue correcta), se llama al método `GetResponse` que devuelve el resultado del método `ExecuteService`.

En la figura 6.9, se muestra el ejemplo de la operación `modificarCestaCompra`. Para esta operación se llama al método `ExecuteService` pasándole como parámetros:

- el nombre de la operación en OlivaNOVA: `modificarCestaCompra`;
- los argumentos de la operación: `cestaCompra`.

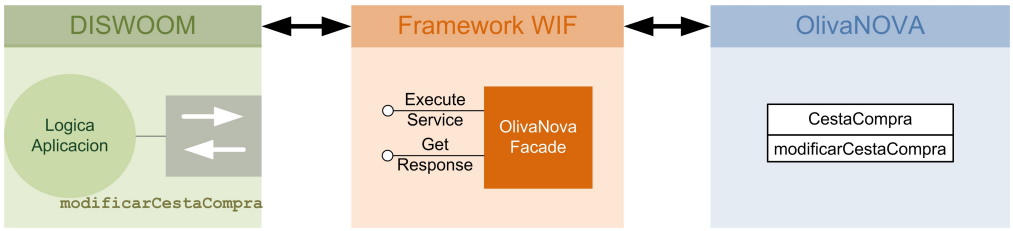

Figura 6.9: Implementación de la estrategia ESGR: `ExecuteService` junto con `GetResponse`

Esta estrategia utiliza dos métodos del framework WIF:

1. Al invocar al método **ExecuteService** se le pasan como parámetros el nombre de la clase, el método de la clase a ejecutar y los parámetros de dicho método.
2. En la invocación del método **GetResponse** no se le pasa ningún parámetro pero espera como respuesta un documento XML.

```
frameworkWIF.executeService(nombreClase.nombreMetodo,
   parametrosMetodo);
return frameworkWIF.getResponse();
```

En los siguientes subapartados se presenta con detalle la implementación de cada uno de los grupos funcionales presentados en el capítulo 5.

6.3.1 Lógica de la Aplicación

Como se ha comentado en el apartado 5.2, en la página 89, las operaciones de este grupo se obtienen del *modelo de requisitos* y su identificación dependen de la aplicación a desarrollar. Al igual que su identificación, su implementación también depende de la aplicación a desarrollar. A continuación se presenta la implementación de las operaciones de este grupo. Para ello, se definen dos

reglas y se sigue una de las estrategias descritas en el apartado anterior, estableciendo qué método/s de OlivaNOVA implementan cada una de las operaciones identificadas en este grupo funcional.

Reglas de transformación

En este apartado se presentan dos reglas para la implementación de las operaciones de la *Lógica de la Aplicación*. La primera regla identifica los parámetros de la operación a implementar, y la segunda regla identifica operación que se va a utilizar del framework WIF y de OlivaNOVA.

La *regla IDC*, definida en la página 95, describe cómo definir los parámetros de las operaciones identificadas. Para ello y con la ayuda de la *Descripción* de la tarea seleccionada, se identifica qué clase y qué método del *modelo de objetos* implementa la tarea. La siguiente regla asume los parámetros del método identificado en la regla 5.2.1, como parámetros de la operación del servicio Web.

REGLA PO: Parámetros de la Operación

Entrada: Salida de la *regla IDC*

Dado el método de la clase (identificado siguiendo la *regla IDC*), se obtienen los parámetros de entrada y salida de la operación.

Salida: Cabecera de la operación

Una vez identificado el método del modelo de objetos que implementa la operación del servicio Web (a partir a la descripción de la tarea) y dado que todas las clases del modelo de objetos están implementadas en el código generado por OlivaNOVA, en la siguiente regla se define qué estrategia, de las presentadas en el apartado anterior, se utiliza para "enganchar" las operaciones de este grupo funcional con la implementación generada por OlivaNOVA utilizando el framework WIF.

Para este grupo funcional, donde se van a llamar a operaciones del modelo

de objetos, las estrategias que se pueden utilizar son:

- La *estrategia ES*, que utiliza el método `ExecuteService` del framework WIF, si se desea llamar a un método específico de la capa de lógica (generada por OlivaNOVA) y no se espera respuesta por parte del mismo.
- La *estrategia ESGR*, que utiliza los métodos `ExecuteService` y `GetResponse` del framework WIF, si se desea llamar a un método específico de la capa de lógica (generada por OlivaNOVA) y se espera respuesta por parte del mismo.

Por lo tanto, se define la siguiente regla:

REGLA OMO: Operación del Modelo de Objetos

Entrada: Estrategias de implementación utilizando el Framework WIF

La implementación de la operación del servicio Web se realiza utilizando la operación del Framework WIF `ExecuteService`, y dependiendo de si se espera o no respuesta se va a utilizar una estrategia u otra.

Regla OMOa: Si el método obtenido (y por tanto la operación del servicio Web) no tiene ningún parámetro de salida, implica que sólo se va a utilizar la operación `ExecuteService`. Por lo tanto se va a seguir la *estrategia ES*.

Regla OMOb: Si el método obtenido (y por tanto la operación del servicio Web) tiene algún parámetro de salida, implica que además de la operación `ExecuteService` también se va a utilizar la operación del Framework WIF `GetResponse`. Por lo tanto se va a seguir la *estrategia ESGR*.

Salida: Estrategia que implementa la operación (*estrategia ES* o *estrategia ESGR*).

En el caso de estudio, un ejemplo de la *regla OMOa* es la operación `borrarCestaCompra` y de la regla *regla OMOb* es la operación `anyadirProductoCestaCompra`. En la figura 6.10 se muestra gráficamente la descripción para la

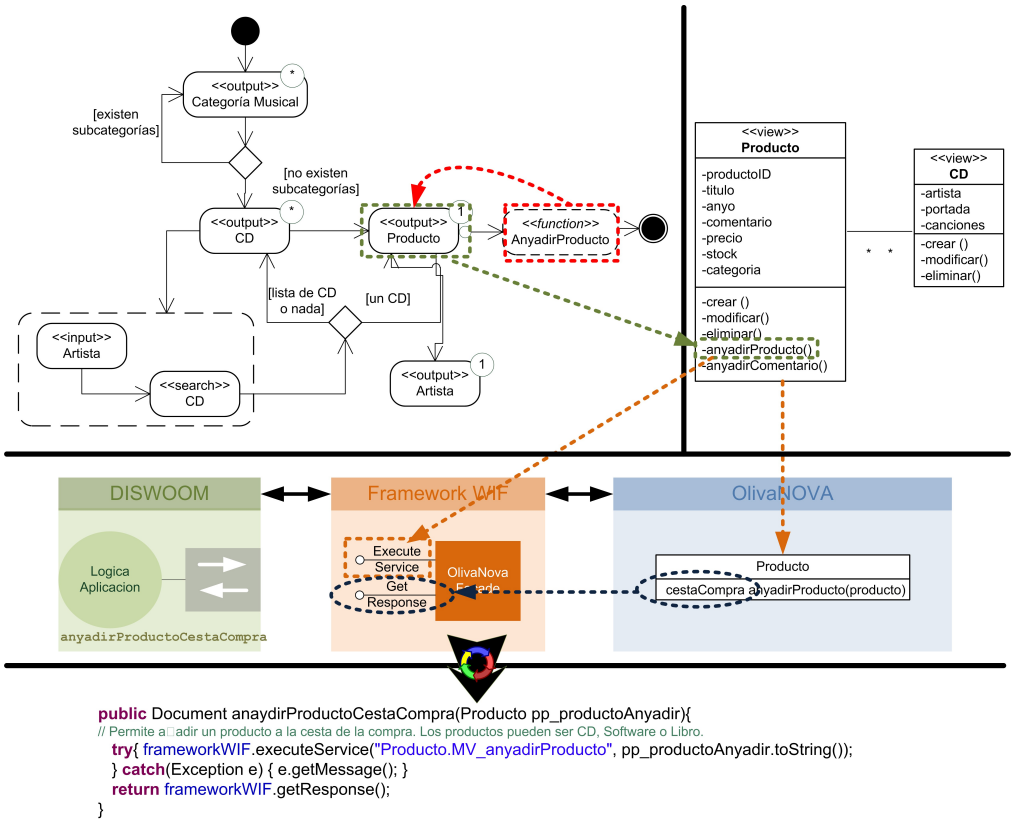

Figura 6.10: Generación de código para la operación `anyadirProductoCestaCompra`

operación `anyadirProductoCestaCompra` siguiendo las reglas anteriormente comentadas:

- A partir de la descripción de la tarea, se identifica la clase `Producto` y el método `anyadirProducto` del modelo de objetos que implementa la operación. La identificación se realiza siguiendo los pasos descritos en la *regla IDC* (ver página 95).

- Siguiendo la *regla PO*, se construye la signatura de la operación con los parámetros de entrada y salida (utilizando también la *regla IDC*). En el ejemplo de la figura 6.10, se encuentra como parámetro de entrada `producto` y como parámetro de salida `cestaCompra`.

- La *regla OMO* construye el cuerpo de la operación. En el ejemplo, al tener parámetro de salida implica que además de la operación `ExecuteService` también se va a utilizar la operación del Framework WIF `GetResponse` (siguiendo la *estrategia ESGR*).

Implementación de las reglas de transformación

A continuación se muestran los algoritmos que implementan las reglas anteriores (`regla PO` y `regla OMO`) en el lenguaje de transformación *Modelo-A-Texto* MOFScript. Los algoritmos muestran el código que se escribe en el fichero Java. Estos algoritmos están formados por:

- Entrada: Estos algoritmos necesitan que se le pasen como entrada una serie de parámetros desde los algoritmos 1 y 2 (páginas 98 y 99 respectivamente).
  - `ps_nombreOperacion`: nombre de la operación a implementar
  - `ps_Entrada`: parámetros de entrada para la operación
  - `ps_clase`: clase que implementa la operación
  - `ps_metodo`: nombre del método que invoca la operación
- Nomenclatura:
  - `dc.Class, c`: clase del modelo de clases
  - `c.Operation, o`: operación de la clase
  - `o.Argument, a`: argumento de la operación

El algoritmo 12 implementa la *regla PO*, que se corresponde con la identificación de los parámetros de las operaciones. Primero, se recorren las clases del modelo de objetos buscando el método que se llama igual que la operación identificada y se toman los parámetros de este método como parámetros de la operación del servicio Web a publicar.

Algoritmo 12 Parámetros de la Operación

```
dc.Class->forEach(c:oows.Class){
  c.Operation->forEach(o:oows.Operation | o.ref.equals
    (ls_nombreOperacion)){
    ls_clase = c.id
    ls_metodo = o.id
    o.Argument->forEach(a:oows.Attribute){
      if (ls_Entrada.size() > 0) {
        ls_Entrada = ls_Entrada + ',' + a.ref
        } else { ls_Entrada = a.ref } } } }
```

La `regla OMO` se desglosa en dos algoritmos: uno por la `regla OMOa` y el otro por la `regla OMOb`. Ambos algoritmos escriben el código Java que realiza las llamadas al framework WIF para implementar las operaciones del servicio Web. Primero se presenta el algoritmo 13, que implementa la estrategia ES (`regla OMOa`).

Algoritmo 13 Estrategia ES para la *Lógica de la Aplicación*

```
 tasksRequirements.Task::operacionConSalida(ps_nombreOperacion:String,
     ps_Entrada:String, ps_clase:String, ps_metodo:String){
  <% public Document %>ps_nombreOperacion<%(%>ps_Entrada<%){ %>
       <% try{
              frameworkWIF.executeService(" %>ps_clase<%.%>ps_metodo<%",
                  %>ps_Entrada <%);
          } catch(Exception e){e.getMessage();}
          return frameworkWIF.getResponse(); }%>
}//operacionConSalida
```

A continuación se presenta el algoritmo 14 que implementa la *estrategia ESGR* `regla OMOb`.

Algoritmo 14 Estrategia ESGR para la *Lógica de la Aplicación*

```
tasksRequirements.Task::operacionSinSalida(ps_nombreOperacion:String,
 ps_Entrada:String, ps_clase:String, ps_metodo:String){
  <% public int %>ps_nombreOperacion<%(%>ps_Entrada<%){ %>
       <% Boolean lb_token = Boolean.FALSE;
          try{
              lb_token = frameworkWIF.executeService(" %>ps_clase<%.%>
                  ps_metodo<%", %>ps_Entrada <%);
          } catch(Exception e){e.getMessage();}
          if (lb_token) {return 0;} else {return -1;} }%>
}//operacionSinSalida
```

Ejemplo del caso de estudio

Como ejemplo de la ejecución del algoritmo 13 está la operación `borrarCestaCompra`. Esta operación se implementa siguiendo la regla *regla OMOa*, que utiliza la *"estrategia ES: **`ExecuteService`**"*, porque no espera respuesta (solamente si la operación se ha realizado con éxito o no).

```
public int borrarCestaCompra(Producto pp_productoABorrar){
  //Permite eliminar un producto o todos los productos que se
     encuentran en la cesta.
  Boolean lb_token = Boolean.FALSE;
  try{
    lb_token = frameworkWIF.executeService("Producto.
       MV_borrarCestaCompra", pp_productoABorrar.toString());
  } catch(Exception e) { e.getMessage(); }
  if (lb_token) {return 0;} else {return -1;} }
```

Como ejemplo de la ejecución del algoritmo 14 se encuentra la operación `anyadirProductoCestaCompra`. Esta operación se implementa siguiendo la regla *regla OMOb*, que utiliza la *"estrategia ESGR: **`ExecuteService`** y **`GetResponse`**"* porque espera una respuesta con el resultado obtenido de la ejecución de la operación.

```
public Document anaydirProductoCestaCompra(Producto
   pp_productoAnyadir){
  // Permite añadir un producto a la cesta de la compra. Los
     productos pueden ser CD, Software o Libro.
  try{
    frameworkWIF.executeService("Producto.MV_anyadirProducto",
       pp_productoAnyadir.toString());
  } catch(Exception e) { e.getMessage(); }
return frameworkWIF.getResponse(); }
```

En ambos ejemplo se ha ejecutado también el algoritmo 12 (*regla PO*) para la identificación de los parámetros.

6.3.2 Gestión de Usuarios

Como se ha comentado en el apartado 5.3, en la página 102, las operaciones de este grupo se obtienen si existe la necesidad de autenticar usuarios en el

sistema (si existe un *rol de tipo registrado* en el *modelo de usuarios*). Aunque para la identificación de las operaciones se ha utilizado el *modelo de usuarios* de OOWS, para su implementación se van a utilizar las clases del *modelo de objetos* de OO-Method. Esto se debe a que en OO-Method los roles aparecen en el propio modelo de objetos, y por lo tanto el código generado por OlivaNOVA incluye tanto la funcionalidad básica para cada uno de los roles, como aquella funcionalidad específica que haya sido modelada. Los métodos básicos que existen para los roles en OlivaNOVA son los de creación, modificación y destrucción de instancias, además de la modificación de la contraseña. Por lo tanto, se aprovecha esta implementación para las operaciones de este grupo.

Una característica de OlivaNOVA, es que no permite el mantenimiento dinámico de roles en el sistema. Esto se debe a que la creación de un nuevo rol significaría la creación de una nueva clase en el modelo de objetos de OO-Method y por lo tanto la lógica asociada generada por OlivaNOVA debería volver a ser generada [3]. Aunque el no crear dinámicamente roles pueda parecer una desventaja, es una práctica muy común en algunos sistemas (sobre todo aquellos sistemas que deban estar amparados por la LOPD [117, 118]).

Reglas de transformación

Como las operaciones de este grupo no dependen de la aplicación a desarrollar, sino que la implementación es siempre la misma (en el caso de que exista algún usuario de tipo registrado), no se ha definido ninguna regla. El algoritmo implementará directamente el código que corresponda en cada caso.

Implementación de las reglas de transformación

A continuación se muestra el algoritmo que implementa las operaciones de este grupo funcional. El algoritmo implementa el código de las operaciones si estas han sido identificadas en el algoritmo 3. Como ejemplo, se muestra

[3] Actualmente se está estudiando una aproximación que permita incorporar nuevos roles en el sistema y así permitir modificar dinámicamente los privilegios de los usuarios en tiempo de ejecución

la implementación de la primera de las operaciones (`iniciarSesion`) que se encuentra dentro de la función `construirSesiones`.

Algoritmo 15 Implementación de la Gestión de Usuarios

```
oows.UserRol::construirSesiones(){
 <%  public Document iniciarSesion(String ps_Rol, String ps_Usuario,
         String ps_Contrasenya){
     try{
         frameworkWIF.executeService(ps_Rol + ".MV_AgentValidation",
             ps_Usuario+","+ps_Contrasenya);
     } catch(Exception e) { e.getMessage(); }
     return frameworkWIF.getResponse(); }
```

Ejemplo del caso de estudio

A continuación, se presentan, a modo de ejemplo, alguna de las operaciones identificadas en el apartado 5.3.2 (una operación por cada estrategia utilizada). Para ellas se detalla el código Java que las implementa, indicando para cada caso qué estrategia se sigue:

- `iniciarSesion`: esta operación utiliza la *"estrategia ESGR:* ***`ExecuteService`*** *y* ***`GetResponse`***" porque espera una respuesta con el resultado obtenido de la ejecución de la operación.

```
public Document iniciarSesion(String ps_Rol, String ps_Usuario,
    String ps_Contrasenya){
  //Permite iniciar la sesión de un usuario.
  try{
    frameworkWIF.executeService(ps_Rol + ".MV_AgentValidation",
        ps_Usuario+","+ps_Contrasenya);
  } catch(Exception e) { e.getMessage(); }
  return frameworkWIF.getResponse(); }
```

- `cerrarSesion`: esta operación utiliza la *"estrategia ES:* ***`ExecuteService`***" porque no espera una respuesta.

```
public int cerrarSesion(String ps_Sesion){
  //Permite cerrar la sesión de un usuario.
  Boolean lb_token = Boolean.FALSE;
  try{
    lb_token = frameworkWIF.executeService("Sesion.
       MV_CloseSesion", ps_Sesion);
  } catch(Exception e) { e.getMessage(); }
  if (lb_token) {return 0;} else {return -1;} }
```

- `buscarUsuariosDeRol`: esta operación utiliza la *"estrategia QR: **Query-Related**"* porque recupera la población de una entidad relacionada.

```
public Document buscarUsuariosDeRol(String ps_Rol, String
   ps_RolID){
  //Busca los usuarios de un rol.
  return frameworkWIF.queryRelated(ps_Rol, "nombre", ps_RolID,"
     UsuariosAsignados"); }
```

6.3.3 Recuperación de Información

Como se ha comentado en el apartado 5.4, las operaciones de este grupo se obtienen de los *contextos navegacionales* del *modelo de navegación*. Al igual que el diseño depende de la aplicación a desarrollar, su implementación también es dependiente. Por ello, se definen cuatro reglas y se siguen las estrategias descritas al inicio del apartado, estableciendo qué método/s de OlivaNOVA implementa cada una de las operaciones identificadas en este grupo funcional.

Reglas de transformación

En este apartado se presentan cuatro reglas para la implementación de las operaciones de la *Recuperación de Información*. Cada una de las reglas se corresponde con una forma de recuperar información: toda la población o una instancia en concreto, la población indexada, la población que cumple una condición o una búsqueda utilizando un filtro.

Tanto las operaciones de OlivaNOVA como los métodos del framework WIF (como se comenta en el apartado 6.3), se centran en la recuperación de población, por lo que tiene métodos específicos para recuperar la información. Además, tres de las estrategias (las *estrategias QP, QBI* y *QR*) se basan es estos métodos:

- La *estrategia QP*, que utiliza el método `QueryPopulation` del framework WIF. Se utiliza esta estrategia siempre que se recupere la población completa de una entidad o se realice una búsqueda sobre la misma.
- La *estrategia QBI*, que utiliza el método `QueryById` del framework WIF. Se utiliza esta estrategia si se tiene el identificador de la instancia a recuperar.
- La *estrategia QR*, que utiliza el método `QueryRelated` del framework WIF. Se utiliza esta estrategia si se recuperan las clases relacionadas de una entidad.

Por tanto, las reglas para definir la implementación de las operaciones de este grupo son las siguientes.

La primera regla define los pasos a seguir para implementar la operación `Recuperar`*`NombreUAI`*.

REGLA IRNU: Implementación de `Recuperar`*`NombreUAI`*

Entrada: Salida de la *regla RNUAI*

Dado un contexto navegacional, por cada UAI se genera una operación de tipo `recuperar`*`NombreUAI`*. Esta operación recupera los datos de la UAI (*regla RNUAI* en la página 116). La implementación de este tipo de operaciones se realiza en tres pasos:

1. Se recuperan una o todas las instancias de la clase directora (dependiendo si se indica un identificador o no) utilizando la *estrategia QBI:* ***`QueryById`*** si solo se desea recuperar 1 o la *estrategia QP:* ***`QueryPopulation`*** si se desean recuperar todas.

2. Se recuperan las instancias de las clases relacionadas con las instancias recuperadas en el primer paso. Para ello, se utiliza la *estrategia QP:* ***`QueryRelated`***.

3. Finalmente se recuperan las instancias de las clases relacionadas para cada una de las instancias recuperadas en el segundo paso. Para ello, también se utiliza la *estrategia QP:* ***`QueryRelated`*** (este paso es igual que el paso 2, por lo que se puede reutilizar su implementación).

Salida: Implementación de las operaciones del tipo `Recuperar`*`NombreUAI`*

En la figura 6.11 se muestra la estrategia de implementación a seguir para la operación `recuperarProducto`. En este ejemplo se observan los tres pasos comentados: (1) recuperar la/s instancia/s de producto/s;, (2) a continuación, recuperar los DVDs, CDs y libros relacionados con la/s instancia/s recuperadas en el paso anterior; y (3) finalmente recuperar el artista de cada CD y el autor de cada libro.

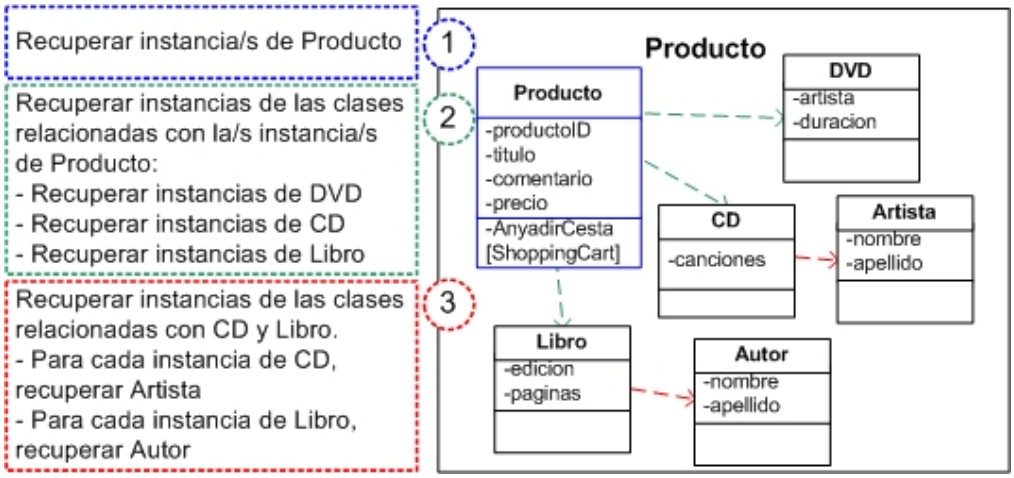

Figura 6.11: Aplicación de la regla IRNU en la operación `recuperarProducto`

La segunda regla define los pasos a seguir para implementar la operación `Indexar`*`NombreIndice`*.

REGLA IINI: Implementación de `Indexar`*`NombreIndice`*

Entrada: Salida de la *regla INI*

Dado un contexto navegacional, por cada índice se genera una operación de tipo `indexar`*`NombreIndice`* (*regla INI* en la página 117). Para las operaciones que recuperan la información de un índice, se aplica el índice y se recupera la población de la clase directora. Para ello, se utiliza la *estrategia QP:* *`QueryPopulation`*.

Salida: Implementación de las operaciones del tipo `Indexar`*`NombreIndice`*

Por ejemplo, para la operación `indexarProductoIndex`, la *regla INII* instanciada en el ejemplo sería: recuperar para todas las instancias de productos su título y su precio.

La tercera regla define los pasos a seguir para implementar la operación `RecuperarPoblacion`*`NombreCondicion`*.

REGLA IRPNC: Implementación de `RecuperarPoblacion`*`Nombre`*

Entrada: Salida de la *regla RPNC*

Dada una clase, por cada mecanismo de condición se genera una operación de tipo `recuperarPoblacion`*`NombreCondicion`* (ver *regla RPNC* en la página 117). La estrategia de implementación para las operaciones que recuperan la información utilizando un mecanismo de condición es la siguiente:

1. Se recupera la población de la clase directora utilizando la *estrategia QP:* *`QueryPopulation`* y pasándole como último parámetro (correspondiente al filtro) la condición introducida en la etapa de modelado.
2. Se recuperan las instancias de las clases relacionadas con las instancias recuperadas en el primer paso. Para ello, se utiliza la *estrategia QP:* *`QueryRelated`*.

3. Finalmente, se recuperan las instancias de las clases relacionadas para cada una de las instancias recuperadas en el segundo paso (este paso es igual que el paso 2, por lo que se puede reutilizar su implementación).

Salida: Implementación de las operaciones del tipo `RecuperarPoblacion-`*`NombreCondicion`*

Como se ha comentado en el capítulo 5, no existe un ejemplo en el caso de estudio para este tipo de operación.

A continuación, se presenta la regla para la implementación de los filtros de búsqueda.

REGLA IBNF: Implementación de `Buscar`*`NombreFiltro`*

Entrada: Salida de la *regla BNF*

Dado un filtro de búsqueda, se genera una operación de tipo `buscar`*`NombreFiltro`* (*regla BNF* en la página 118). La estrategia de implementación para las operaciones que recuperan la información de un filtro de búsqueda, es la siguiente:

1. Se recupera la población de la clase directora utilizando la *estrategia QP:* *`QueryPopulation`*, pasándole como parámetro el filtro introducido por el usuario [4].
2. Se recuperan las instancias de las clases relacionadas con las instancias recuperadas en el primer paso. Para ello se utiliza la *estrategia QP:* *`QueryRelated`*.
3. Se recuperan las instancias de las clases relacionadas para cada una de las instancias recuperadas en el segundo paso (este paso es igual que el paso 2, por lo que se puede reutilizar su implementación).

Salida: Implementación de las operaciones del tipo `Buscar`*`NombreFiltro`*

Siguiendo con el ejemplo del contexto navegacional `Producto`, la regla IBNF se instancia para la operación `buscarProducto` de la siguiente forma: recuperar las instancias de productos que cumplan la condición indicada en el filtro de búsqueda; a continuación, recuperar los DVDs, CDs y libros relacionados con las instancias recuperadas en el primer paso; y finalmente recuperar el artista de cada CD, y el autor de cada libro.

Implementación de las reglas de transformación

A continuación, se presentan los cuatro algoritmos que implementan las reglas anteriormente presentadas (*reglas IRNU, IINI, IBNF e IRPNC*).

Estos algoritmos necesitan que se le pasen como entrada una serie de parámetros desde los algoritmos 4, 5, 6 y 7:

- `ps_NombreUAI`: nombre de la UAI
- `ps_NombreIndice`: nombre del índice del contexto definido en la UAI
- `ps_NombreCondicion`: nombre de la condición de población definido en la UAI
- `ps_NombreFiltro`: nombre del filtro definido en la UAI
- `ps_ClasePpal`: nombre de la clase principal de la UAI
- `ps_ParametrosPpal`: nombre de los atributos de la clase principal de la UAI
- `ps_ClaseSec`: nombre de las clases secundarias de la UAI
- `ps_ParametrosSec`: nombre de los atributos de las clases complementarias de la UAI

Antes de empezar a ver la implementación de las reglas, se presenta la implementación del paso 2 (el paso 3 tiene la misma implementación que el paso 2 por lo que se reutiliza su implementación). Estos pasos son compartidos

por las *reglas IRNU, IRPNC y IBNF* (definidas en las páginas 163, 165 y 166 respectivamente).

El algoritmo 16 recibe como parámetros de entrada el nombre de la UAI, el nombre de la clase principal, el nombre de las clases secundarias y sus parámetros. Este algoritmo recupera la información (los atributos declarados en los parámetros) de las clases relacionadas (clases secundarias) con la clase principal.

Algoritmo 16 `RecuperarPASO2`

```
oows.AIU::RecuperarPaso2(ps_NombreUAI:String, ps_ClasePpal:String, pl_ClaseSec:List,
   pl_ParametrosSec:List){
  ps_ClasePpal<% lp_%>ps_ClasePpal<%AProcesar = (%>ps_ClasePpal<%) ld_%>ps_ClasePpal
   <%.getFirstChild();%>
       pl_ClaseSec->forEach(ls_ClaseSec) {
         <% Document ld_%>ls_ClaseSec<% = null; %> }
       <% while (lp_%>ps_ClasePpal<%AProcesar != null){
         String ls_%>ps_ClasePpal<%ID = lp_%>ps_ClasePpal<%AProcesar.get%>
           ps_ClasePpal<%_Id(); %>
       var ls_Parametros = ''
       pl_ClaseSec->forEach(ls_ClaseSec) {
         ls_Parametros = pl_ParametrosSec.first()
         pl_ParametrosSec.remove(ls_Parametros)
         <%//%>ls_ClaseSec<%
         ld_%>ls_ClaseSec<%.appendChild(frameworkWIF.queryRelated("%>ps_ClasePpal
           <%","%>ls_Parametros<%",  ls_%>ps_ClasePpal<%ID, "%>ls_ClaseSec<%"));
           %> }
       <% lp_%>ps_ClasePpal<%AProcesar = (%>ps_ClasePpal<%) ld_%>ps_ClasePpal
           <%.getElementById(ls_%>ps_ClasePpal<%ID).getNextSibling(); }%>
       pl_ClaseSec->forEach(ls_ClaseSec) {
        <% ld_recuperar%>ps_NombreUAI<%.appendChild(ld_%>ls_ClaseSec<%); %> }
}//RecuperarPaso2
```

Una vez visto cómo se implementa el paso 2, a continuación se presentan los algoritmos que implementan las reglas para la implementación del grupo funcional *Recuperación de Información*.

El primer algoritmo que se presenta implementa la *regla IRNU* (definida en la página 163). Este algoritmo define el código Java que implementa las operaciones de tipo `recuperar`*`NombreUAI`*.

Algoritmo 17 Implementación de `Recuperar`*`NombreUAI`*

```
oows.AIU::RecuperarNombreUAI(ps_NombreUAI:String, ps_ClasePpal:String,
   ps_ParametrosPpal:String, pl_ClaseSec:List, pl_ParametrosSec:List){
  <% public Document recuperar%> ps_NombreUAI <% (String ps_%> ps_ClasePpal <%_Id)
   Document ld_recuperar%>ps_NombreUAI<% = null;
   Document ld_%>ps_ClasePpal<% = null;
   try { //PASO1
       if (ps_%>ps_ClasePpal<%_Id.isEmpty()){ //Recupera toda la población
           ld_%>ps_ClasePpal<%.appendChild(frameworkWIF.queryPopulation("%>
             ps_ClasePpal<%", "%>ps_ParametrosPpal<%")); }
       else{ //Recupera una instancia en concreto
           ld_%>ps_ClasePpal<%.appendChild(frameworkWIF.queryById("%>
             ps_ClasePpal<%", "%>ps_ParametrosPpal<%, ps_%>ps_ClasePpal<%_Id)); }
       ld_recuperar%>ps_NombreUAI<%.appendChild(ld_%>ps_ClasePpal<%); %>
       if (pl_ClaseSec.size() > 0){  <% //PASO 2   %>
         self.RecuperarPaso2(ps_NombreUAI, ps_ClasePpal, pl_ClaseSec,
           pl_ParametrosSec) }<%
   } catch(Exception e) {e.getMessage(); }
   return ld_recuperar%>ps_NombreUAI<%; } %>
}//RecuperarNombreUAI
```

A continuación, la regla que se presenta es la *regla IINI* (definida en la página 165). Este algoritmo define el código Java que implementa las operaciones de tipo `indexar`*`NombreIndice`*.

Algoritmo 18 Implementación de `Indexar`*`NombreIndice`*

```
oows.AIU::indexarNombreIndice(ps_NombreIndice:String, ps_ClasePpal:String,
   ps_ParametrosPpal:String){
  <%public Document indexar%>ps_NombreIndice<%(){
   Document ld_indexar%>ps_NombreIndice<% = null;
   try { //PASO1
       ld_indexar%>ps_NombreIndice<%.appendChild(frameworkWIF.queryPopulation
         ("%>ps_ClasePpal<%", "%>ps_ParametrosPpal<%"));
   } catch(Exception e) { e.getMessage(); }
   return ld_indexar%>ps_NombreIndice<%; } %>
}//indexarNombreIndice
```

El algoritmo 19 presenta la implementación de la *regla IRPNC* (definida en la página 165). Este algoritmo reutiliza el algoritmo 16 que implementa el paso 2, y define el código Java que implementa las operaciones de tipo `recuperarPoblacion`*`NombreCondicion`*.

Algoritmo 19 Implementación de RecuperarPoblacion*NombreCondicion*

```
oows.AIU::recuperarPoblacionNombreCondicion(ps_NombreCondicion:String,
    ps_ParametrosPpal:String, pl_ClaseSec:List,  pl_ParametrosSec:List){
   <%public Document recuperarPoblacion%>ps_NombreCondicion<%(){
    Document ld_recuperarPoblacion%>ps_NombreCondicion<% = null;
    try { //PASO 1
      ld_recuperarPoblacion%>ps_NombreCondicion<%.appendChild(frameworkWIF.
        queryPopulation("%>ps_NombreCondicion<%", "%>ps_ParametrosPpal<%"));
        %> if (pl_ClaseSec.size() > 0){
          <% //PASO 2   %>
          self.RecuperarPaso2(ps_NombreCondicion, ps_NombreCondicion,
            pl_ClaseSec, pl_ParametrosSec) }<%
    } catch(Exception e) { e.getMessage(); }
    return ld_recuperarPoblacion%>ps_NombreCondicion<%; } %>
}   //recuperarPoblacionNombreCondicion
```

Para terminar, se presenta el algoritmo 20 que es la implementación de la *regla IBNF* (definida en la página 166). Este algoritmo define el código Java que implementa las operaciones de tipo buscar*NombreFiltro*.

Algoritmo 20 Implementación de Buscar*NombreFiltro*

```
oows.AIU::buscarNombreFiltro(ps_NombreFiltro:String, ps_ClasePpal:String,
   ps_ParametrosPpal:String){
  <%public Document buscar%>ps_NombreFiltro<%(){
   Document ld_buscar%>ps_NombreFiltro<% = null;
   try {
       ld_buscar%>ps_NombreFiltro<%.appendChild(frameworkWIF.queryPopulation
         ("%>ps_ClasePpal<%", "%>ps_ParametrosPpal<%"));
   } catch(Exception e) { e.getMessage(); }
   return ld_buscar%>ps_NombreFiltro<%;  } %>
}//buscarNombreFiltro
```

Ejemplo del caso de estudio

A continuación, para cada uno de los algoritmos presentados, se muestra una de las operaciones identificadas en el apartado 5.4.2.

La operación recuperarProducto se implementa siguiendo el algoritmo 17. Para su implementación se utilizan la *estrategia QBI:* ***QueryById*** si solo se desea recuperar 1 instancia o la *estrategia QP:* ***QueryPopulation*** si se desean recuperar todas, y la *estrategia QP:* ***QueryRelated*** para recuperar las instancias relacionadas.

```
public Document recuperarProducto(String ps_productoId){
//Permite recuperar la información todos los productos (si no
   se le pasa ningún parámetro) o de un producto en particular
   (si se pasa su identificador como parámetro.
Document ld_recuperarProducto = null;
Document ld_Producto = null;
try {
//PASO1
if (ps_productoId.isEmpty()){ //Recupera toda la población
 ld_Producto.appendChild(frameworkWIF.queryPopulation("Producto
    ", "productoId,titulo,comentario,precio"));
}
else{ //Recupera una instancia en concreto
 ld_Producto.appendChild(frameworkWIF.queryById("Producto", "
    titulo,comentario,precio", ps_productoId));
}
ld_recuperarProducto.appendChild(ld_Producto);

//PASO 2
Producto lp_ProductoAProcesar = (Producto) ld_Producto.
   getFirstChild();
Document ld_DvdRelacionados = null;
Document ld_CdRelacionados = null;
Document ld_LibroRelacionados = null;
while (lp_ProductoAProcesar != null){
 String ls_ProductoID = lp_ProductoAProcesar.getProductoId();
 //DVD
 ld_DvdRelacionados.appendChild(frameworkWIF.queryRelated("
    Producto", "artista,duracion",  ls_ProductoID, "Ddv"));
 //CD
 ld_CdRelacionados.appendChild(frameworkWIF.queryRelated("
    Producto", "canciones",  ls_ProductoID, "Cd"));
 //Libro
 ld_LibroRelacionados.appendChild(frameworkWIF.queryRelated("
    Producto", "edicion,paginas",  ls_ProductoID, "Libro"));
 lp_ProductoAProcesar = (Producto) ld_Producto.getElementById(
    ls_ProductoID).getNextSibling();
}
ld_recuperarProducto.appendChild(ld_DvdRelacionados);
ld_recuperarProducto.appendChild(ld_CdRelacionados);
ld_recuperarProducto.appendChild(ld_LibroRelacionados);

//PASO 3
Cd lcd_CdAProcesar = (Cd) ld_CdRelacionados.getFirstChild();
Document ld_ArtistasRelacionados = null;
```

```
while (lcd_CdAProcesar != null){
 String ls_CdId = lcd_CdAProcesar.getCdId();
 ld_ArtistasRelacionados.appendChild(frameworkWIF.queryRelated(
    "Cd", "nobmre,apellido", ls_CdId, "Artista"));
 lcd_CdAProcesar = (Cd) ld_CdRelacionados.getElementById(
    ls_CdId).getNextSibling();
}
ld_recuperarProducto.appendChild(ld_ArtistasRelacionados);
//Autores_Libros
Libro ll_LibroAProcesar = (Libro) ld_LibroRelacionados.
   getFirstChild();
Document ld_AutoresRelacionados = null;
while (ll_LibroAProcesar != null){
 String ls_LibroId = ll_LibroAProcesar.getLibroId();
 ld_AutoresRelacionados.appendChild(frameworkWIF.queryRelated("
    Libro", "nombre,apellido",  ls_LibroId, "Autor"));
}
ld_recuperarProducto.appendChild(ld_AutoresRelacionados);
} catch(Exception e)    { e.getMessage(); }
return cp_Producto; }
```

La operación `indexarProductoIndex` se implementa siguiendo el algoritmo 18. Para su implementación se utiliza la *estrategia QP:* ***QueryPopulation*** para recuperar todas las instancias y la *estrategia QP:* ***QueryRelated*** para recuperar las instancias relacionadas.

```
public Document indexarProductoIndex(){
//Esta operación provee un acceso indexado a la población de
   productos por su nombre y precio.
Document ld_indexarProductoIndex = null;
try {
 //PASO1
 //Recupera toda la población para el índice definido
 ld_indexarProductoIndex.appendChild(frameworkWIF.
    queryPopulation("Producto", "titulo,precio"));
} catch(Exception e) { e.getMessage(); }
 return ld_indexarProductoIndex; }
```

La operación `buscarProducto` se implementa siguiendo el algoritmo 20. Para su implementación se utilizan la *estrategia QP:* ***QueryPopulation*** pasándole un filtro para recuperar todas las instancias que cumplen la condición del filtro, y la *estrategia QP:* ***QueryRelated*** para recuperar las instancias relacionadas.

```
public Document buscarProducto(String ps_Filtro){
//Esta operación filtra el espacio de productos que recupera
   por nombre.
Document ld_buscarProducto = null;
Document ld_Producto = null;
try {
 //PASO1
 //Recupera toda la población
 ld_Producto.appendChild(frameworkWIF.queryPopulation("Producto
    ", "productoId,titulo,comentario,precio", "titulo like" +
    ps_Filtro));
 ld_buscarProducto.appendChild(ld_Producto);

 //PASO 2
 Producto lp_ProductoAProcesar = (Producto) ld_Producto.
    getFirstChild();
 Document ld_DvdRelacionados = null;
 Document ld_CdRelacionados = null;
 Document ld_LibroRelacionados = null;
 while (lp_ProductoAProcesar != null){
  String ls_ productoID = lp_ProductoAProcesar.getProductoId();
  //DVD
  ld_DvdRelacionados.appendChild(frameworkWIF.queryRelated("
     Producto", "artista,duracion",  ls_ProductoID, "Ddv"));
  //CD
  ld_CdRelacionados.appendChild(frameworkWIF.queryRelated("
     Producto", "canciones",  ls_ProductoID, "Cd"));
  //Libro
                        ld_LibroRelacionados.appendChild(
                           frameworkWIF.queryRelated("Producto"
                           , "edicion,paginas",  ls_ProductoID,
                            "Libro"));
  lp_ProductoAProcesar = (Producto) ld_Producto.getElementById(
     ls_ProductoID).getNextSibling();
 }
 ld_buscarProducto.appendChild(ld_DvdRelacionados);
 ld_buscarProducto.appendChild(ld_CdRelacionados);
 ld_buscarProducto.appendChild(ld_LibroRelacionados);

 //PASO 3
 //Artistas_CD
 Cd lcd_CdAProcesar = (Cd) ld_CdRelacionados.getFirstChild();
 Document ld_ArtistasRelacionados = null;
 while (lcd_CdAProcesar != null){
  String ls_CdId = lcd_CdAProcesar.getCdId();
```

```
 ld_ArtistasRelacionados.appendChild(frameworkWIF.queryRelated
    ("Cd", "nobmre,apellido", ls_CdId, "Artista"));
 lcd_CdAProcesar = (Cd) ld_CdRelacionados.getElementById(
    ls_CdId).getNextSibling();
}
ld_buscarProducto.appendChild(ld_ArtistasRelacionados);
//Autores_Libros
Libro ll_LibroAProcesar = (Libro) ld_LibroRelacionados.
   getFirstChild();
Document ld_AutoresRelacionados = null;
while (ll_LibroAProcesar != null){
 String ls_LibroId = ll_LibroAProcesar.getLibroId();
 ld_AutoresRelacionados.appendChild(frameworkWIF.queryRelated(
    "Libro", "nombre,apellido",  ls_LibroId, "Autor"));
 ll_LibroAProcesar = (Libro) ld_LibroRelacionados.
    getElementById(ls_LibroId).getNextSibling();
}
ld_buscarProducto.appendChild(ld_AutoresRelacionados);

} catch(Exception e) {  e.getMessage(); }
 return ld_buscarProducto; }
```

6.3.4 Soporte a la Navegación

Como se ha comentado en el apartado 5.5, las operaciones del grupo *Soporte a la Navegación* se obtienen del *modelo de navegación* (mapas y contextos navegacionales). Al contrario que su diseño, la implementación de este grupo depende de la aplicación a desarrollar, por lo que un cambio en el *modelo de navegación* significa tener que volver a generar la implementación de este servicio.

Reglas de transformación

En este apartado se presentan las reglas para la implementación de las operaciones de este grupo. Estas operaciones son propias de OOWS y no se puede utilizar ninguna de las estrategias presentadas al principio del apartado 6.3 por no estar presentes en la implementación de OlivaNOVA. La herramienta OOWS Suite tampoco ofrece una implementación que pueda ser utilizada

para estas operaciones, por lo tanto, se van implementar en las siguientes reglas.

La primera regla explica cómo implementar la operación que recupera todos los enlaces de *exploración*.

REGLA IREE: Implementación de `recuperarEnlaceExploracion`

Entrada: Mapa navegacional

- Se recorren todos los contextos del mapa.
- Se devuelven los contextos cuya navegación sea de tipo *Exploración*.

Salida: Implementación de la operación `recuperarEnlaceExploracion`

Por ejemplo, para el caso de estudio del comercio electrónico, la operación `recuperarEnlacesExploracion` devuelve los contextos `Home`, `Categorías Producto`, `Cesta Compra`, `Clientes` y `Vendedores`.

A continuación, se presenta la regla para implementar la operación de recuperación de enlaces de *secuencia*.

REGLA IRES: Implementación de `recuperarEnlaceSecuencia`

Entrada: Mapa navegacional

- Se recorren todos los contextos del mapa navegacional.
- Por cada par de contextos cuya navegación sea de tipo *secuencia*, se devuelve el contexto origen y el destino.

Salida: Implementación de la operación `recuperarEnlaceSecuencia`

Para el caso de estudio del comercio electrónico, la operación `recuperar-EnlaceSecuencia` dado un contexto origen devuelve los siguientes contextos (primero está el contexto origen y tras la flecha el/los contexto/s destino/s):

- *Categorías Productos* ⇒ *Producto*
- *Producto* ⇒ *Cesta Compra* y *Comentarios Clientes*
- *Clientes* ⇒ *Comentarios Clientes*

Finalmente, se presenta la regla que implementa la operación de recuperación de enlaces de *servicio*.

REGLA IRESV: Implementación de `recuperarEnlaceServicio`

Entrada: Mapa navegacional

- Se recorren todos los contextos del mapa.
- Se busca cada contexto que tenga un enlace de servicio, se devuelve el servicio y el contexto destino al que se llega tras su ejecución.

Salida: Implementación de la operación `recuperarEnlaceServicio`

Siguiendo con el caso de estudio, se encuentra un enlace de servicio en la operación `anyadirCesta` que tras su ejecución redirige la navegación al contexto *Cesta Compra*.

Implementación de las reglas de transformación

A continuación se muestran los algoritmos para generar el código de las reglas anteriormente presentadas (*reglas IREE, IRES y IRESV*).

El algoritmo 21 implementa la *regla IREE*. Este algoritmo, junto con el algoritmo 8, genera el código de las operaciones de tipo `recuperarEnlaceExploracion`. Devuelve en la lista `ls_EnlacesExploracion`, todos los enlaces de exploración que haya definidos en el modelo navegacional.

Algoritmo 21 Implementación de `recuperarEnlaceExploracion`

```
    if (not coleccionEnlaceExploracion.isEmpty()) {
     ls_EnlacesExploracion = ''
<% public String recuperarEnlacesExploracion() {
    return "%>
          coleccionEnlaceExploracion->forEach(ee) {
            if (ls_EnlacesExploracion.size() > 0) {
              ls_EnlacesExploracion = ls_EnlacesExploracion + ','
                 + ee
            }else { ls_EnlacesExploracion = ee }  }
        ls_EnlacesExploracion <%"; } %> }
```

El segundo algoritmo implementa la *regla IRES*. Este algoritmo, junto con el algoritmo 9, genera el código de las operaciones de tipo `recuperarEnlaceSecuencia`. Dado un contexto, este algoritmo devuelve el enlace de secuencia al cual se accede.

Algoritmo 22 Implementación de `recuperarEnlaceSecuenciarecuperarEnlaceSecuencia`

```
if (not coleccionEnlaceSecuencia.isEmpty()){
     li_numeroEnlaces = 0
        <% public recuperarEnlacesSecuencia(String ps_contexto) {%>
          coleccionEnlaceSecuencia->forEach(es) {
            li_numeroEnlaces = li_numeroEnlaces + 1
            if (li_numeroEnlaces == 1){
    <% if (ps_contexto == "%> es.substringBefore(" ---> ")<%") {
        return "%> es.substringAfter(" ---> ")<%"; %>
            } else { <% } else { if (ps_contexto == "%>
              es.substringBefore(" ---> ")<%") {
        return "%> es.substringAfter(" ---> ")<%"; %>  } }
    <% } else { return "-1"%>
        li_numeroEnlaces->forEach(n){ <% }%> } <% } %> }
```

Finalmente se presenta el algoritmo 23 que implementa la *regla IRESV*. Este algoritmo, junto con el algoritmo 10, devuelve el enlace al cual se accede después de la ejecución de un servicio.

Algoritmo 23 Implementación de `recuperarEnlaceServicio`

```
ur.NavigationalNode->forEach(no:oows.NavigationalOperation) {
 li_numeroEnlaces = 0
 <% public String recuperarEnlaceServicio(String ps_servicio){ %>
   li_numeroEnlaces = li_numeroEnlaces + 1
   if (li_numeroEnlaces == 1){
     <% if (ps_servicio == "%> no.Operation.id <%") {
    return "%> no.ServiceLink.TargetContextLink.id <%"; %>
    } else {
<% } else { if (ps_servicio == "%> no.Operation.id <%") {
    return "%> no.ServiceLink.TargetContextLink.id <%"; %> }
<% } else { return "-1"%>
    li_numeroEnlaces->forEach(n){
    <% }%>   } <% } %> }
```

Ejemplo del caso de estudio

A continuación, para cada una de las operaciones identificadas en el apartado 5.5.2, se detalla el código Java que las implementa. Cabe recordar que este grupo funcional no utiliza ninguna implementación generada por OlivaNOVA.

- `recuperarEnlacesExploracion`

```
public String recuperarEnlacesExploracion(){
  return "Home,Categorias␣Productos,Cesta␣Compra,Clientes,␣
     Vendedores";
}
```

- `recuperarEnlacesSecuencia`

```
public String recuperarEnlaceSencuencia(String ps_contexto){
  if (ps_contexto == "Categorias␣Productos") { return "producto
     "; }
  else { if (ps_contexto == "Producto") { return "cestaCompra,␣
     comentariosClientes"; }
    else { if (ps_contexto == "clientes") { return "
       comentariosClientes"; }
      else { return "-1"; }  }  }        }
```

- `recuperarEnlaceServicio`

```
public String recuperarEnlaceOperacion(String ps_servicio){
  if (ps_servicio == "anyadirCesta") { return "cestaCompra"; }
  else { return "-1"; } }
```

6.3.5 Soporte a la Presentación

Como se ha comentado en el apartado 5.6, las operaciones de este grupo se obtienen de los contextos navegacionales del *modelo de navegación* y los *modelos de presentación*. Al igual que su identificación, su implementación también depende de la aplicación a desarrollar.

Reglas de transformación

En este apartado se presentan las reglas para la implementación de las operaciones del grupo *Soporte a la Presentación*. Estas operaciones, al igual que las operaciones del grupo de *Soporte a la Navegación*, son propias de OOWS y por lo tanto no se puede utilizar ninguna de las estrategias presentadas al inicio del apartado (en el punto 6.3), al no estar presentes en la implementación de OlivaNOVA. OOWS Suite tampoco ofrece una implementación que pueda ser utilizada.

La primera regla presenta la estrategia de implementación de la operación `presentacionDeInformacion`.

REGLA IPIN: Implementación de `presentacionDeInformacion`

Entrada: Modelo de presentación, una operación de las diseñadas anteriormente

- Se invoca una operación a través de `presentacionDeInformacion`.
- Se accede a la plantilla definida por la herramienta OOWS Suite a partir del modelo de presentación.
- A continuación, `presentacionDeInformacion` llama a la operación y le asocia, al resultado obtenido, la plantilla CSS.
- Finalmente, `presentacionDeInformacion` devuelve un documento html con el XML y la plantilla CSS.

Salida: Implementación de la operación `presentacionDeInformacion`

La figura 6.12 muestra gráficamente esta estrategia.

Figura 6.12: Estrategia de la operación `presentacionDeInformacion`

También es posible que el consumidor de la operación `presentacionDeInformacion` indique las propiedades de presentación (paginación, ordenación

y presentación de instancias) que desea sean aplicadas al resultado. Si estas propiedades se indican, entonces no se tienen en cuenta las propiedades definidas en el modelo de presentación. De esta forma, se facilita a los consumidores de los servicios Web, que adapten la presentación de las propiedades a sus preferencias.

Un concepto importante para comprender la generación de la siguiente regla (para las operaciones de tipo `presentacionDe`*`NombreContexto`*), es que una página Web se considera como una agregación de un conjunto de áreas de contenido [119]. Las áreas que se utilizan para generar el código de esta operación son:

- *Área de Información* (ver figura 6.13, zona 1): presenta los datos y la funcionalidad a la que pueden acceder los usuarios. Esta área se implementa usando las operaciones que dan soporte a la recuperación de información (presentadas en el grupo *Recuperación de Información*).
- *Área de Navegación* (ver figura 6.13, zona 2): provee los enlaces a las páginas Web que están disponibles a los usuarios. Esta área se implementa usando las operaciones que dan soporte a los enlaces navegacionales (presentadas en el grupo *Soporte a la Navegación*).
- *Área de Acceso estructurado* (ver figura 6.13, zona 3): provee a los usuarios con los mecanismos de acceso a la información, como mecanismos de búsqueda, los cuales facilitan el acceso de la información. Esta área se implementa con las operaciones que soportan la búsqueda o los índices (presentadas en el grupo *Recuperación de Información*).

De este modo, para generar las operaciones de `presentacionDe`*`NombreContexto`*, como por ejemplo `presentacionDeProducto`, se han de generar las operaciones que implementen las áreas comentadas. Cada área se implementa llamando a las operaciones de los grupos funcionales *Recuperación de Información* y *Soporte a la Navegación*. Por ejemplo, el *Área de Información* utiliza las operaciones `recuperarProducto` y `recuperarProductosSimilares`.

La figura 6.13 muestra la página Web creada como resultado de la operación `presentacionDeProducto`. La información recuperada desde la UAI *Producto* se muestra como registros, en grupos de 4 y ordenada por nombre en orden

ascendente. También se muestra la información de la UAI *Productos Similares* y un campo para buscar por nombre. Además, se muestran los enlaces a las páginas Web que se pueden alcanzar desde este contexto (a través de los enlaces de *exploración* y de *secuencia*).

Figura 6.13: Zonas de la operación `presentacionDeProducto`

Por lo tanto, la operación `presentacionDeProducto` se implementa utilizando diferentes operaciones que implementas cada una de las áreas (ver figura 6.14):

- `Área de Información`, implementada utilizando las operaciones de *Recuperación de Información* de cada una de las UAIs que conforman el contexto *Producto*: `recuperarProducto` y `recuperarProductosSimilares`.
- `Área de Navegación`, implementada utilizando las operaciones de *Soporte a la Navegación*: `recuperarEnlaceExploracion`, `recuperarEnlaceSecuencia` y `recuperarEnlaceServicio`.
- `Área de Acceso Estructurado`, implementada utilizando las operaciones de *Recuperación de Información* que representan los filtros de búsquedas y los índices: `indexarProductoIndex` y `buscarProducto`.

Esto se resume en la siguiente regla.

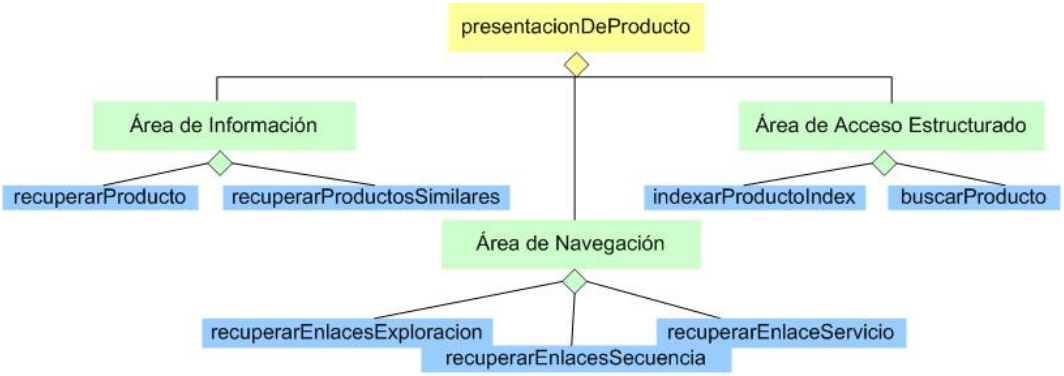

Figura 6.14: Agregación de operaciones para `presentacionDeProducto`

REGLA IPNC: Implementación de `presentacion`*`NombreContexto`*

Entrada: Salida de la *regla PNC* Dado un contexto, se genera una operación de tipo `presentacionDe`*`NombreContexto`* (ver *regla PNC* en la página 134). La implementación de este tipo de operaciones se realiza a través de sus tres áreas:

- *Área de Información*:
  - Se recorren los contextos del mapa navegacional.
  - Para cada contexto, se recorren las UAI que lo forman y se llama a la operación que las implementa.
- *Área de Navegación*:
  - Se implementa la llamada a la operación `recuperarEnlaceExploracion`
  - Se implementa la llamada a la operación `recuperarEnlaceSecuencia`, pasándole como parámetro de entrada el nombre del contexto.
  - Si alguna de las UAIs tiene un enlace de servicio, se implementa la llamada a la operación `recuperarEnlaceServicio`, pasándole como parámetro de entrada el nombre del servicio.

- *Área de Acceso Estructurado*:
  - Para cada índice definido en el contexto, se realiza la llamada a la operación que lo implementa.
  - Para cada condición de población definida en el contexto, se realiza la llamada a la operación que la implementa.
  - Para cada filtro de búsqueda definido en el contexto, se realiza la llamada a la operación que lo implementa.

Salida: Implementación de las operaciones del tipo `presentacion`*`NombreContexto`*

Una vez vistas las reglas, a continuación se presentan los algoritmos que las implementan.

Implementación de las reglas de transformación

El primer algoritmo implementa la *regla IPIN* (definida en la página 180). Este algoritmo, junto con el algoritmo 11, implementa la operación `presentacion-DeInformacion`.

Algoritmo 24 Implementación de `presentacionDeInformacion`

```
//presentacionDeInformacion
<%
    public HTMLDocument presentacionDeInformacion (String ps_operacion,
       String[] ps_parametrosOperacion, String ps_paginacion,
       String ps_orden, String ps_instancias){ %>
    <%
       HTMLDocument lhd_presentacionDeInformacion = null;
       try { lhd_presentacionDeInformacion.open();
           Document ld_datos = ps_operacion(ps_parametrosOperacion);
           HTMLTableElement lte_tabla = null;
           Node ln_dato = ld_datos.getFirstChild();
           while (ln_dato != null){
               lte_tabla.appendChild(ln_dato);
               ln_dato = ld_datos.getNextSibling(); }
           lhd_presentacionDeInformacion.appendChild(lte_tabla);
           lhd_presentacionDeInformacion.close();
       } catch(Exception e){ e.getMessage(); }
       return lhd_presentacionDeInformacion; }%>
```

El segundo algoritmo implementa la *regla IPNC* (definida en la página 183). Este algoritmo, junto con el algoritmo 11, implementa las operaciones del tipo `presentacionDe`*`NombreContexto`*.

Algoritmo 25 Implementación de `presentacionDe`*`NombreContexto`*

```
//presentacionDeNombreContexto
 self.NavigationalModel.UserRol->forEach(ur:oows.UserRol)  {
   ur.NavigationalNode->forEach(nn:oows.NavigationalNode)  {
     SPList.add("presentacionDe"+nn.id)
   <% public HTMLDocument presentacionDe%> nn.id <%(){
         HTMLDocument lhd_%>nn.id<% = null;
         try { lhd_%>nn.id<%.open();
           lhd_%>nn.id<%.setTitle("Generado con DISWOOM");
           Document ld_datos = swRI.recuperar%>nn.id<%(null);
           HTMLTableElement lte_tabla = null;
           Node ln_dato = ld_datos.getFirstChild();
           while (ln_dato != null){
             lte_tabla.appendChild(ln_dato);
             ln_dato = ld_datos.getNextSibling(); }
           lhd_%>nn.id<%.appendChild(lte_tabla);
           String ls_Enlaces = swSN.recuperarEnlacesExploracion();
           lhd_%>nn.id<%.writeln(ls_Enlaces);
           lhd_%>nn.id<%.close();
         } catch(Exception e){ e.getMessage(); }
         return lhd_%>nn.id<%; } %>
```

Ejemplo del caso de estudio

A continuación, para cada uno de los algoritmos presentados, se muestra una de las operaciones identificadas en el apartado 5.6. Cabe recordar que este grupo funcional no utiliza ninguna implementación generada por OlivaNOVA.

La primera operación, `presentacionDeInformacion`, se implementa siguiendo el algoritmo 24.

```
public HTMLDocument presentacionDeInformacion (String
   ps_operacion, String[] ps_parametrosOperacion, String
   ps_paginacion, String ps_orden, String ps_instancias){
  //Devuelve los datos de un Servicio Web con formato
  HTMLDocument lhd_presentacionDeInformacion = null;
  try {
    lhd_presentacionDeInformacion.open();
    lhd_presentacionDeInformacion.setTitle(‘‘Generado con
       DISWOOM’’);
```

```
    Document ld_datos = ps_operacion(ps_parametrosOperacion);
    HTMLTableElement lte_tabla = null;
    Node ln_dato = ld_datos.getFirstChild();
    while (ln_dato != null){
      lte_tabla.appendChild(ln_dato);
      ln_dato = ld_datos.getNextSibling(); }
    lhd_presentacionDeInformacion.appendChild(lte_tabla);
    lhd_presentacionDeInformacion.close();
  } catch(Exception e){ e.getMessage(); }
  return lhd_presentacionDeInformacion; }
```

La operación `presentacionDeProducto` se implementa siguiendo el algoritmo 25.

```
public HTMLDocument presentacionDeProducto(String ps_Filtro){
  //Devuelve una página completa de un contexto
  HTMLDocument lhd_Producto = null;
  try {
    lhd_Producto.open();
    Document ld_datos = swRI.recuperarProducto(null);
    HTMLTableElement lte_tabla = null;
    Node ln_dato = ld_datos.getFirstChild();
    while (ln_dato != null){
      lte_tabla.appendChild(ln_dato);
      ln_dato = ld_datos.getNextSibling(); }
    lhd_Producto.appendChild(lte_tabla);
    String ls_Enlaces = swSN.recuperarEnlacesExploracion();
    lhd_Producto.writeln(ls_Enlaces);
swRI.buscarProducto(ps_Filtro);
    lhd_Producto.close();
  } catch(Exception e){ e.getMessage(); }
  return lhd_Producto; }
```

6.4 Conclusiones

En este capítulo se ha presentado un método para la generación automática de código para los servicios Web a partir de los modelos conceptuales de OO-Method / OOWS. Para ello se utilizan los métodos del framework WIF, desarrollado como parte de la herramienta OOWS Suite, que da soporte a OOWS, y la implementación generada por OlivaNOVA, herramienta que da soporte

a OO-Method.

Para cada grupo funcional, se ha presentado la implementación de las operaciones identificadas para el caso de estudio de comercio electrónico (una por cada tipo de operación identificada. La generación de todas las operaciones se encuentra en el capítulo A).

Este método, junto con el presentado en el capítulo 5, forman la estrategia de diseño e implementación de servicios Web presentada en esta tesis. En el siguiente capítulo, se presenta `DISWOOM` (Diseño e Implementación de Servicios Web para OO-Method / OOWS). `DISWOOM` es la herramienta que da soporte a este método, cubriendo tanto el diseño como la implementación de todos los grupos funcionales presentados.

Tabla 6.1: Implementación de los grupos funcionales

Grupo Funcional	Método del Framework WIF				
	QP	QBI	QR	ES	ESGR
Lógica Aplicación	No	No	No	Si	Si
Gestión Usuarios	No	No	Si	Si	Si
Recuperación Información	Si	Si	Si	No	No
Soporte Navegación	No	No	No	No	No
Soporte Presentación	No	No	No	No	No

En la tabla 6.1 se puede ver un resumen de los métodos del *framework WIF* que se utilizan en la implementación de cada uno de los grupos funcionales:

- QP: Implementación utilizando el método `QueryPopulation`.
- QBI: Implementación utilizando el método `QueryById`.

- QR: Implementación utilizando el método `QueryRelated`.
- ES: Implementación utilizando el método `ExecuteService`.
- ESGR: Implementación utilizando los métodos `ExecuteService` y `GetResponse`.

Capítulo 7

DISWOOM: Herramienta para la generación de servicios Web

Una de las contribuciones de esta tesis es DISWOOM (Diseño e Implementación de Servicios Web para OO-Method), una herramienta que implementa las ideas propuestas a lo largo de la tesis. En este capítulo se presenta esta herramienta, junto con las herramientas que le dan apoyo en el proceso de desarrollo de servicios Web.

La generación de servicios Web, siguiendo la metodología dirigida por modelos, está soportada por la herramienta DISWOOM. De acuerdo con el proceso de desarrollo descrito en el capítulo 4 (página 78), el proceso se divide en dos actividades: (1) la primera actividad se encarga de diseñar los servicios Web (capítulo 5); y (2) en la segunda se genera la implementación de las operaciones que conforman el servicio Web (capítulo 6).

El capítulo comienza con una descripción general de la herramienta. A continuación, se introducen las herramientas utilizadas en el proceso de desarrollo de los servicios Web. En el tercer apartado se presentan, paso a paso, los resultados de estas herramientas (utilizados por la herramienta DISWOOM como entrada para generar los servicios Web). Finalmente, se presentan las conclusiones.

7.1 Introducción

Inicialmente, en la Ingeniería del Software se utilizaban los modelos con el único propósito de documentar la aplicación para los clientes. Posteriormente, estos modelos fueron relegados a un plano secundario dentro del proceso de desarrollo de software. Sin embargo, aproximaciones como el Desarrollo Dirigido por Modelos (MDD) han cambiado esta realidad. Esta aproximación convierte a los modelos en ciudadanos de primera clase dentro del proceso de desarrollo del software. La principal ventaja de esta aproximación es que las aplicaciones software pueden ser generadas automáticamente a partir de un conjunto de modelos. Además, la madurez alcanzada por las tecnologías y estándares permite que MDD sea una realidad. Un ejemplo de la madurez de estas herramientas es el Proyecto de Modelado de Eclipse (EMP).

El capítulo presenta la herramienta `DISWOOM`, la cual ha sido desarrollada haciendo uso del entorno Eclipse. Para su desarrollo se han utilizado un conjunto de plugins para Eclipse (lenguajes para la transformación de modelos), que han sido combinados para dar soporte al proceso de desarrollo presentado en esta tesis.

Eclipse comenzó siendo el entorno para el desarrollo Java de IBM. Actualmente, Eclipse es la plataforma base para muchas otras tecnologías y proyectos por su potente estructura modular y su naturaleza abierta. Muchos de los plug-ins están relacionados con el desarrollo de software, pero no necesariamente utilizando Java.

Eclipse está organizado en un conjunto de proyectos temáticos, que han evolucionado en proyectos más concretos. El Proyecto de Modelado de Eclipse (*Eclipse Modeling Project*) es el primero de estos proyectos, que unifica el modelado relacionado con proyectos y los plug-ins que han sido desarrollados por la comunidad Eclipse (otros plug-ins de modelado están siendo desarrollados por terceras partes desde que Eclipse es una plataforma abierta). Para el desarrollo de la herramienta `DISWOOM` ha sido utilizado uno de los proyectos que se encuentran dentro del Proyecto de Modelado Eclipse: MOFScript [15].

Se ha utilizado MOFScript como lenguaje de transformaciones para implementar las transformaciones de *Modelo-A-Texto*, vistas en los capítulos 5 y 6. Las herramientas de MOFScript, que implementan el lenguaje MOFScript,

están incluidas en el proyecto de Eclipse GMT.

MOFScript ha sido elegido por las siguientes razones:

- Es un lenguaje específicamente diseñado para la transformación de modelos en ficheros de texto, que es la propuesta de esta tesis.
- Está estructurado usando transformaciones y reglas. En el trabajo se proponen una serie de reglas para transformar los modelos a código.
- Trata directamente con descripciones de metamodelos (ficheros Ecore). Tanto el metamodelo de OO-Method / OOWS como el de Requisitos se encuentran en este formato.
- Las transformaciones de MOFScript pueden ser ejecutadas directamente desde el entorno de Eclipse, lo cual simplifica mucho la herramienta.

7.2 Herramientas en el proceso de desarrollo de servicios Web

La generación de servicios Web, siguiendo el método dirigido por modelos presentado en esta tesis, está soportado por la herramienta `DISWOOM`. De acuerdo con el proceso de desarrollo descrito en el capítulo 4, este proceso se divide en cuatro fases: *Especificación de Requisitos*, *Modelado Conceptual*, *Generación de código* y ***servicios Web***. La figura 7.1 resume las actividades que se llevan a cabo y las herramientas usadas en cada fase.

La primera actividad es la *Especificación de Requisitos* (apartado 4.2 en la página 58). En esta actividad se construye el modelo de requisitos y está soportada por la herramienta RE Tool [82].

En la segunda actividad, el *Modelado Conceptual* (apartado 4.3 en la página 63), se construyen los modelos conceptuales OO-Method (estructural, dinámico y funcional) y OOWS (usuario, navegacional y presentación). Esta actividad está soportada por dos aplicaciones: OlivaNOVA [91] para los modelos OO-Method y OOWS Suite para los modelos OOWS [12].

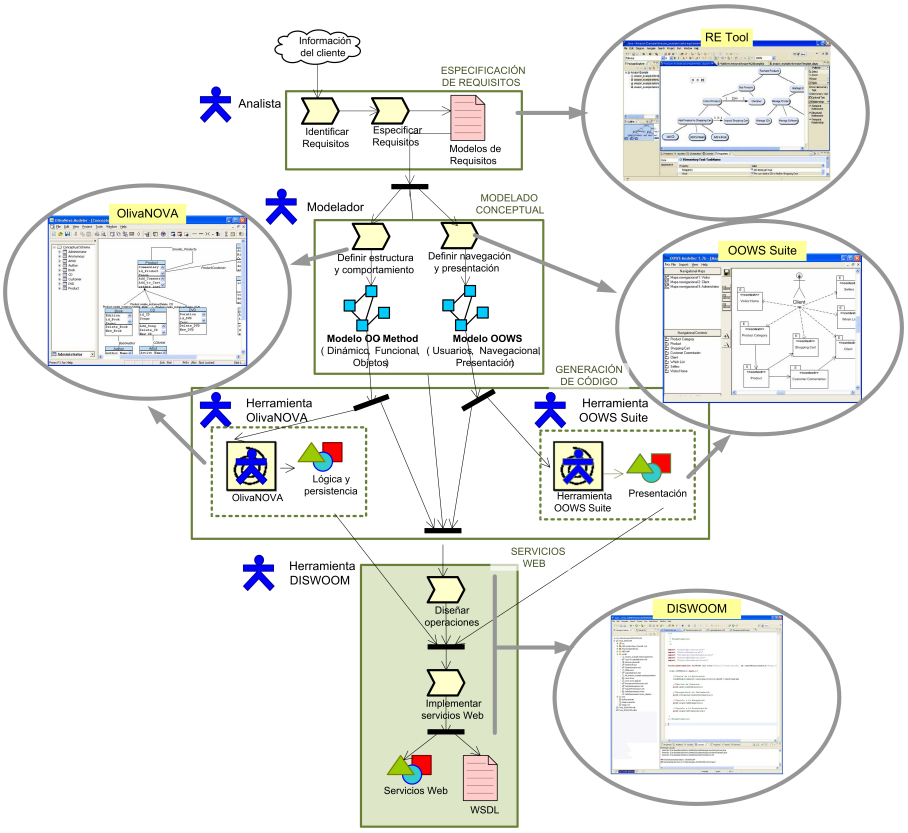

Figura 7.1: Herramientas que dan soporte al proceso de desarrollo de servicios Web

La tercera actividad, la *Generación de Código* (apartado 4.4 en la página 73), genera el código de la aplicación a través de las dos herramientas que dan soporte a OO-Method [92] y a OOWS.

Finalmente, la actividad de *servicios Web* (apartado 4.5 en la página 78, y en los capítulos 5 y 6), diseña e implementa los servicios Web utilizando los modelos y el código generado en las actividades anteriores (ver figura 7.2). La herramienta que da soporte a esta última actividad es DISWOOM. Esta última actividad, junto con la herramienta que le da soporte, son las contribuciones de la presente tesis.

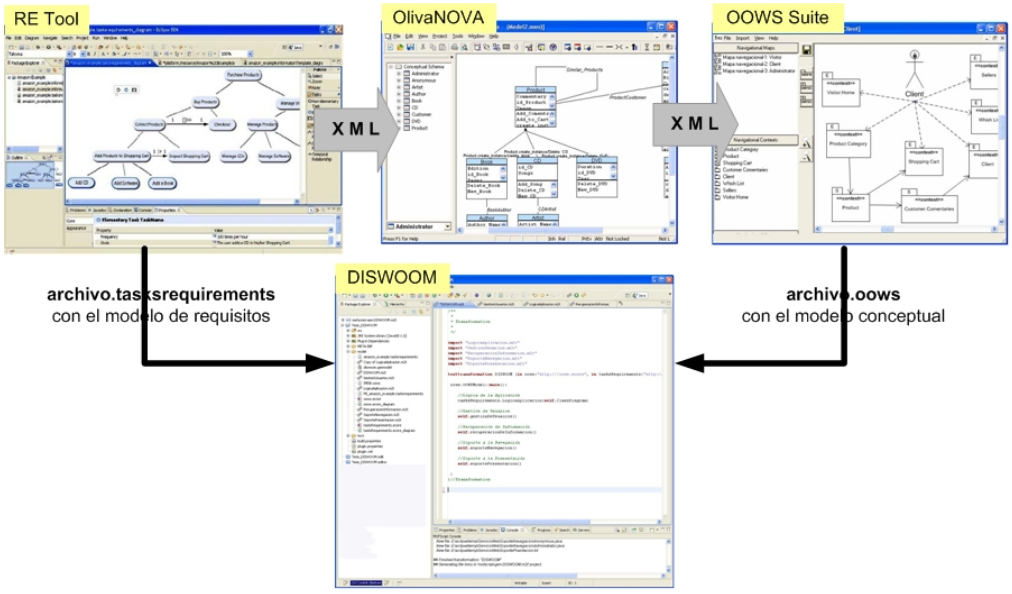

Figura 7.2: Herramientas que dan soporte a DISWOOM

7.3 De modelos conceptuales a servicios Web

De acuerdo con el apartado 7.2, los diferentes pasos que se llevan a cabo para generar servicios Web a partir de los modelos conceptuales son los siguientes:

1. Crear el modelo de requisitos. Para crearlo, se utiliza la herramienta RE TOOL.
2. Crear el modelo conceptual de la aplicación y el código que la implementa. Para ello se utilizan dos herramientas: OLIVANOVA y OOWS SUITE.
3. Generar los servicios Web. Para generarlos, se utiliza la herramienta DISWOOM.

A continuación, se analizan los resultados obtenidos en cada paso cuando se utiliza la herramienta correspondiente para el caso de estudio. Además, se explica cómo el resultado obtenido en cada paso se utiliza como entrada para el siguiente.

7.3.1 Paso 1: Construcción del Modelo de Requisitos

En este paso se analizan y documentan las necesidades funcionales que deberán ser soportadas por el sistema a desarrollar. El resultado obtenido del primer paso es un modelo de requisitos (apartado 4.2 en la página 58). Este modelo se ha creado en la herramienta RE TOOL.

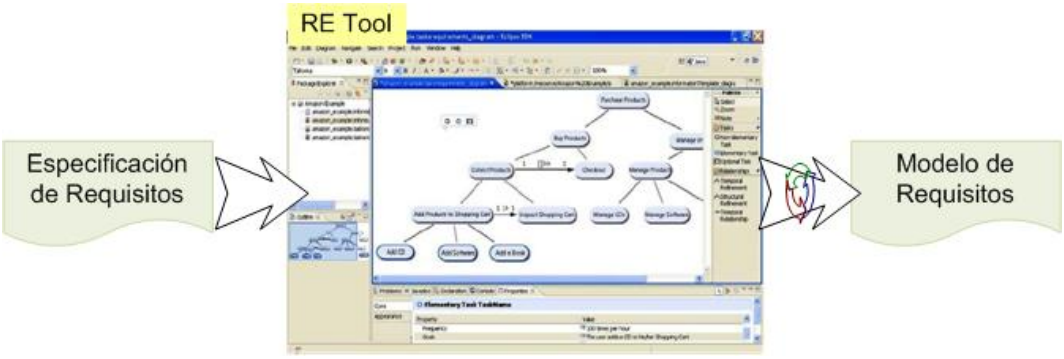

Figura 7.3: Paso 1: creación del modelo de requisitos

Como se puede observar en la figura 7.3, la entrada de la herramienta es la especificación de requisitos del usuario y la salida es el propio modelo de requisitos, formado por el modelo de tareas y la descripción de las tareas elementales.

7.3.2 Paso 2: Construcción del Modelo Conceptual y Generación de Código

En este paso se modela el sistema siguiendo el proceso de desarrollo propuesto en OO-Method / OOWS. Este paso se divide en dos:

2.1. Creación del modelo conceptual (apartado 4.3 en la página 63). Por un lado, se define la estructura y comportamiento (modelos de OO-Method) en la herramienta OLIVANOVA. Por otro, se definen la navegación y presentación (modelos de OOWS) en la herramienta OOWS SUITE.

2.2. Generación de código (apartado 4.4 en la página 73). A partir de los modelos generados en el paso 2.1, se genera el código de la aplicación. Por un lado, se genera la capa de persistencia y lógica a partir de los modelos OO-Method. Por otro, se genera la capa de presentación a partir de los modelos OOWS.

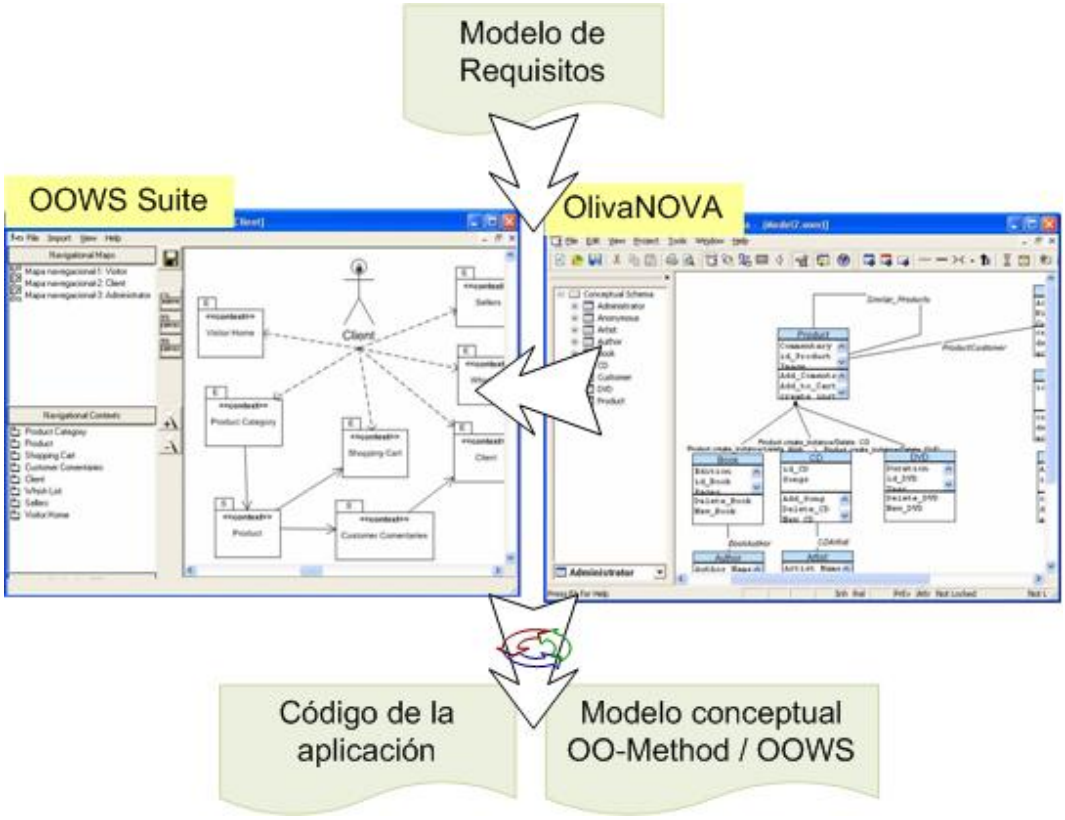

Figura 7.4: Paso 2: creación del modelo conceptual y generación de código

Como se puede observar en la figura 7.4, la entrada de la herramienta es el modelo de requisitos y la salida son el modelo conceptual y el código de la aplicación.

7.3.3 Paso 3: Generación de servicios Web

La herramienta DISWOOM permite identificar los servicios Web a publicar y la generación automática de su implementación mediante un conjunto de transformaciones.

Como se puede observar en la figura 7.5, la entrada de la herramienta es el modelo de requisitos y el modelo conceptual, y la salida son los servicios Web generados.

Figura 7.5: Paso 3: generación de los servicios Web

Como se ha comentado, las entradas para esta herramienta son dos (ver figura 7.6):

- Un fichero *.oows*, con el modelo OO-Method / OOWS.
- Un fichero *.tasksrequirements*, con el modelo de la Especificación de Requisitos.

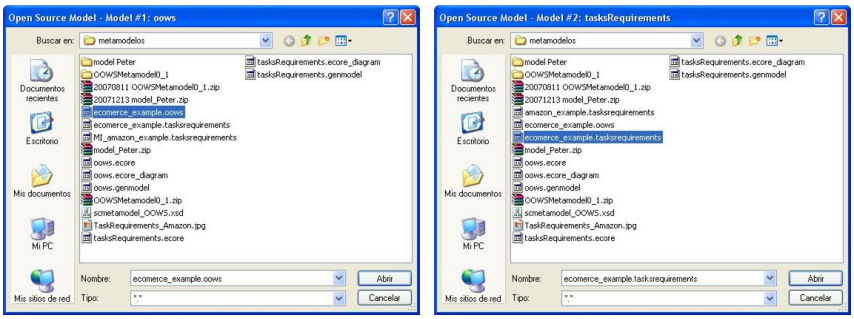

Figura 7.6: Ficheros de entrada

7.4 Conclusiones

En este capítulo, se han presentado las diferentes herramientas utilizadas en el proceso de desarrollo para diseñar e implementar servicios Web siguiendo una aproximación dirigida por modelos. Se han detallado, para cada paso del proceso, las herramientas. Para estas herramientas se han indicado qué resultados producen y cómo estos resultados son la entrada de las otras herramientas.

La herramienta que da soporte a esta tesis, `DISWOOM`, está formada por un conjunto de transformaciones (utilizando MOFScript), para generar el código que implementa los servicios Web a partir de modelos OO-Method / OOWS. Las transformaciones han sido implementadas para (1) identificar las operaciones de los servicios Web e (2) implementar el código Java de cada una de las operaciones.

Capítulo 8

Conclusiones y Trabajos Futuros

En esta tesis se ha introducido un método para la generación automática de servicios Web a partir de modelos conceptuales. Para alcanzar esta meta, se ha desarrollado una solución dentro del campo de la Ingeniería Web. Además, la solución propuesta aplica el enfoque del Desarrollo de Software Dirigido por Modelos (DSDM), donde la definición de transformaciones de modelos permite transformar los modelos que define la aplicación en una implementación específica.

Este último capítulo introduce las conclusiones del trabajo desarrollado en esta tesis. Primero, se presentan las principales contribuciones a la comunidad de Ingeniería Web. A continuación, se comenta tanto el trabajo que se está llevando a cabo, como el trabajo futuro que ha emergido de esta tesis. Finalmente, se citan las publicaciones que han hecho posible el desarrollo de este trabajo.

8.1 Contribuciones

Las principales contribuciones en el ámbito de la Ingeniería Web son:

- Se ha definido un proceso completo para llevar a cabo el desarrollo de

servicios Web. Este proceso comienza tras la fase de modelado (la fase donde el sistema se representa en términos de un conjunto de modelos) y se extiende hasta la fase de generación (la fase que aplica el conjunto de reglas para obtener la implementación de los servicios Web).

- Se han definido un conjunto de reglas para obtener: (1) el diseño de los servicios Web y (2) su implementación. Estas reglas guían al modelador en el diseño e implementación de los servicios Web.
- A partir de estas reglas se han definido un conjunto de transformaciones de modelos utilizando MOFScript. Estas transformaciones constituyen la correspondencia entre los modelos de OO-Method / OOWS y su representación software basada en servicios Web.
- Se ha desarrollado la herramienta *DISWOOM* que implementa las ideas presentadas en este trabajo. La herramienta ha sido desarrollada aplicando las últimas tendencias en el campo de MDA. *DISWOOM* ha sido desarrollada usando herramientas incluidas en el entorno de desarrollo de Eclipse, y se ha utilizado MOFScript para implementar las transformaciones definidas en la propuesta.

8.2 Trabajo Futuro

El trabajo desarrollado tiene continuidad en varias áreas de trabajo dentro del grupo de investigación, que se pueden abordar en el futuro. Estas áreas son:

- **Operaciones independientes del dominio**. Algunas de las operaciones presentadas se pueden generalizar, de manera que no sean dependientes del dominio. Estas operaciones son los cuatro tipos de operación identificada en el grupo *Recuperación de Información* (`recuperar`*`NombreUAI`*, `indexar`*`NombreIndice`*, `recuperarPoblacion`*`NombreCondicion`* y `buscar`*`NombreFiltro`*), y las operaciones `presentacionDe`*`NombreContexto`* del *Soporte a la presentación.*
- **Modelado de servicios Web**. La propuesta presentada puede ampliarse generando, tras el diseño de los servicios Web y previo a su implementación, el modelado de estos servicios. Para ello, se pretende utilizar

el *Modelo de Servicios* presentado en [45] e instanciar cada uno de los servicios Web con sus operaciones en este nuevo modelo.

- **Extensiones del código generado**. Como se ha comentado en el capítulo 6, han habido 2 decisiones importantes a la hora de implementar los servicios Web: Java y SOAP. Un trabajo interesante sería extender la implementación para cubrir otros lenguajes de implementación, como por ejemplo C#, y también presentar una implementación de servicios Web basados en REST. Actualmente se está trabajando en un metamodelo y en la integración de servicios REST [120].

- **Directorio centralizado de usuarios**. Crear un directorio centralizado de usuarios, donde distintas aplicaciones Web comparten un perfil de usuario para poder realizar adaptatividad navegacional basada en este.

- **Variabilidad en la composición de Servicios Web**. Los procesos de negocio definen el conjunto de tareas que permite conseguir los objetivos de una organización. Aunque los objetivos de una organización suelen ser estables, el cómo conseguirlos puede cambiar dependiendo del contexto actual de la organización. Para dar soporte a esta naturaleza cambiante de los procesos de negocio, se propone dar soporte tanto a nivel de modelado como de ejecución siguiendo una estrategia dirigida por modelos. Por un lado, a nivel de modelado se propone especificar la variabilidad de los procesos de negocio (procesos especificados en BPMN) de forma separada utilizando el lenguaje CVL (Common Variability Language). Por otro lado, a nivel de ejecución, se generará de forma automática (mediante transformaciones *Modelo-A-Texto*) especificaciones en BPEL (o extensiones a ésta como VxBPEL) que recojan toda la variabilidad especificada en los modelos.

Parte de esta tesis se está materializando en el trabajo diario realizado en Indra Software Labs.

- Se están utilizando las operaciones del grupo *Gestión de Usuarios* para la elaboración de la gestión de identidad en algunos proyectos como ITACA (Innovación Tecnológica Administrativa para Centros y Alumnos) [121].

- Las guías utilizadas para el diseño de los servicios Web se están aplicando en el día a día de este proyecto.

- Se pretende reutilizar las reglas implementadas en `DISWOOM` para generar código siguiendo las reglas de negocio definidas para las aplicaciones desarrolladas en Indra Software Labs.

Existen dos temas relacionados con los servicios Web que no han sido tratados en la tesis y que son una buena línea de continuación dentro de Indra Software Labs:

- La evolución de los servicios Web [97], es un tema en el cual Indra Software Labs está muy interesado.
- La seguridad de los servicios Web, es un tema que se está tratando actualmente y que sería interesante profundizar en él.

8.3 Publicaciones

El trabajo desarrollado en esta tesis ha sido publicado en los siguientes capítulos de libros, conferencias y congresos:

Congresos Internacionales

- **Marta Ruiz**, Pedro Valderas, Vicente Pelechano. *Applying a Web Engineering Method to Design Web Services*. Service-Oriented Computing - ICSOC. Springer-Verlag. *Lecture Notes in Computer Science* (ISSN 0302-9743), volumen 3826, 576 - 581 pp. (2005)
- **Marta Ruiz**, Pedro Valderas, Victoria Torres, Vicente Pelechano. *A Model Driven Approach to Design Web Services in a Web Engineering Method*. Conference on Advanced Information Systems Engineering Forum - CAiSE Forum. FEUP ediçoes. Proceedings of CAiSE' 05 (ISBN 972-752-078-2), 183 - 188 pp. (2005)
- Victoria Torres, Vicente Pelechano, **Marta Ruiz**, Pedro Valderas. *A Model Driven Approach for the Integration of External Functionality in Web Applications. The Travel Agency System*. ICWE org. Proceedings

of The ICWE 2005 Workshop on Model-Driven Web Engineering (ISBN 1-74128-105-9), 1 - 11 pp. (2005)

- **Marta Ruiz**, Vicente Pelechano, Oscar Pastor. *Designing Web Services for Supporting User Tasks: A Model Driven Approach*. Advances in conceptual modeling: Theory and practice - COSS. Springer-Verlag. *Lecture Notes in Computer Science* (ISSN 0302-9743), volumen 4231, 193 - 202 pp. (2006)

- **Marta Ruiz**, Vicente Pelechano. *Model Driven Design of Web Service Operations using Web Engineering Practices*. Emerging Web Services Technology - WEWST (ISBN 978-3-7643-8447-0), 83 - 100 pp. (2006)

- **Marta Ruiz**, Pedro Valderas, Vicente Pelechano. *Providing Methodological Support to Incorporate Presentation Properties in the Development of Web services*. E-Commerce and Web Technologies - ECWEB. Spinger-Verlag. *Lecture Notes in Computer Science* (ISSN 0302-9743), volumen 4655, 139 -148 pp. (2007)

Congresos Nacionales

- **Marta Ruiz**, Ausiàs Armesto, Vicente Pelechano. *Hacia la Generación de Aplicaciones Web en Arquitecturas SOA. Una aproximación basada en MDA*. Jornadas de Ingeniería de Software y Base de Datos - JISBD. Servicio de Publicaciones Universidad de Málaga. (ISBN 84-688-8983-0), 531 - 538 pp. (2004)

- **Marta Ruiz**, Pedro Valderas, Victoria Torres, Vicente Pelechano. *Una Aproximación Dirigida por Modelos para el Diseño de Servicios Web en el ámbito de un Método de Ingeniería Web*. I Jornadas Científico-Técnicas en Servicios Web - JSWEB. Actas del CEDI 2005 (ISBN 84-9732-455-2), 133 - 140 pp. (2005)

- **Marta Ruiz**, Francisco Valverde, Vicente Pelechano. *Desarrollo de servicios Web en un método de generación de código dirigido por modelos*. II Jornadas Científico-Técnicas en Servicios Web - JSWEB. (ISBN 84-690-2398-5), 14 - 22 pp. (2006)

Capítulos de libros

- Vicente Pelechano, **Marta Ruiz**, Joan Fons, Pedro Valderas. *Desarrollo de Aplicaciones WEB basadas en Servicios WEB XML. Un Caso Práctico*. Avalon Programming Solutions. Avances en Comercio Electrónico (ISBN 84-607-5827-3), volumen 7, 99 - 118 pp. (2002)
- Pedro J. Valderas Aranda, **Marta Ruiz Server**, Victoria Torres Bosch, Vicente Pelechano Ferragud. *Especificación de Requisitos en el Desarrollo de Aplicaciones Web*. Avalon Programming Solution. Tendencias en el Desarrollo de Aplicaciones Web (ISBN 84-688-7872- 3), volumen 8, 101 - 118 pp. (2004)

La tabla 8.1 resume y clasifica las publicaciones de acuerdo a su lugar de publicación.

Tabla 8.1: Resumen de Publicaciones

Categoría	**Número**	**Acrónimo**
Congresos internacionales	6	ICSOC (LNCS), CAiSE Forum, ICWE, COSS (LNCS), WEWST, ECWEB (LNCS)
Congresos nacionales	3	JISBD, JSWEB (2)
Capítulos de libro	2	Avalon Programming Solution (2)
Total	11	

Apéndice A

Un caso de estudio: una aplicación de comercio electrónico

A.1 Introducción

En este apéndice se introduce un caso de estudio donde las contribuciones de esta tesis se ponen en práctica. En este caso de estudio se desarrolla una aplicación de comercio electrónico, como Amazon, mediante servicios Web. Se ha escogido una aplicación de comercio electrónico porque este tipo de aplicaciones se centran tanto en proveer una gran cantidad de información como en proveer funcionalidad compleja para por ejemplo comprar productos. Estas propiedades permiten mostrar las características principales del método para el diseño y la implementación de servicios Web. Se ha utilizado una aplicación de comercio electrónico similar a Amazon para facilitar la comprensión del método, y porque Amazon es una de las aplicaciones de comercio electrónico más utilizadas y conocidas.

Las principales características del caso de estudio son las siguientes:

El caso de estudio es una aplicación de comercio electrónico que so-

porta la compra de productos en línea. Esta aplicación de comercio electrónico, permite a los usuarios (que se conectan inicialmente al sistema como usuarios anónimos) acceder a la información sobre los diferentes productos que existen en el catálogo.

Para cada producto, los usuarios pueden recuperar la información relevante del producto como por ejemplo el nombre, precio y una breve descripción (entre otras características).

La aplicación de comercio electrónico debe proporcionar a los usuarios, una cesta de la compra donde puedan añadir productos. Los usuarios pueden añadir tantos productos como deseen. La cesta de la compra debe estar siempre disponible a los usuarios para permitirles su administración. Los usuarios deben poder acceder a la cesta de la compra para recuperar los productos que han añadido. Además deben poder eliminar o modificar la cantidad de productos que han añadido.

Cuando un producto se añade a la cesta de la compra, el sistema debe actualizar automáticamente las unidades del producto.

Una vez que los usuarios han seleccionado los productos a comprar (añadiéndolos a la cesta de la compra), la aplicación de comercio electrónico debe permitirles crear y enviar una orden de compra. Para hacer esto, los usuarios deben identificarse como clientes registrados en el sistema. Para ser un cliente registrado, deben proporcionar al sistema la siguiente información: su nombre y apellidos, dirección, ciudad donde viven así como el país y el código postal, su teléfono y un nombre de usuario y contraseña. Cuando el sistema pide los datos para autenticar al usuario, este debe introducir el nombre de usuario y la contraseña.

Finalmente, toda la información sobre productos es administrada por el usuario Administrador. Los administradores pueden introducir nuevos productos en el catálogo, modificar la información o eliminar los productos existentes. Además, los administradores pueden administrar la información de los clientes registrados. También pueden crear nuevos clientes, modificar su información e incluso eliminarlos.

A continuación, se presenta cómo se aplica la transformación *Modelo-A-*

Texto a los modelos OO-Method / OOWS que representan el caso de estudio. Se muestra la traza que permite obtener las operaciones y la implementación de los servicios Web.

A.2 Transformación de modelos a servicios Web

En este apartado se presenta la transformación de los servicios Web a partir de los modelos OO-Method / OOWS del caso de estudio.

A.2.1 Lógica de la Aplicación

A continuación, se describe la aplicación del algoritmo propuesto (ver los algoritmos 1 y 2 en la página 98 y 99) sobre el caso de estudio. Este algoritmo se aplica sobre el modelo de tareas de la figura 5.2, para identificar las operaciones a publicar en el servicio Web siguiendo los pasos del algoritmo. Para cada uno de los pasos descritos se enumera la línea del algoritmo.

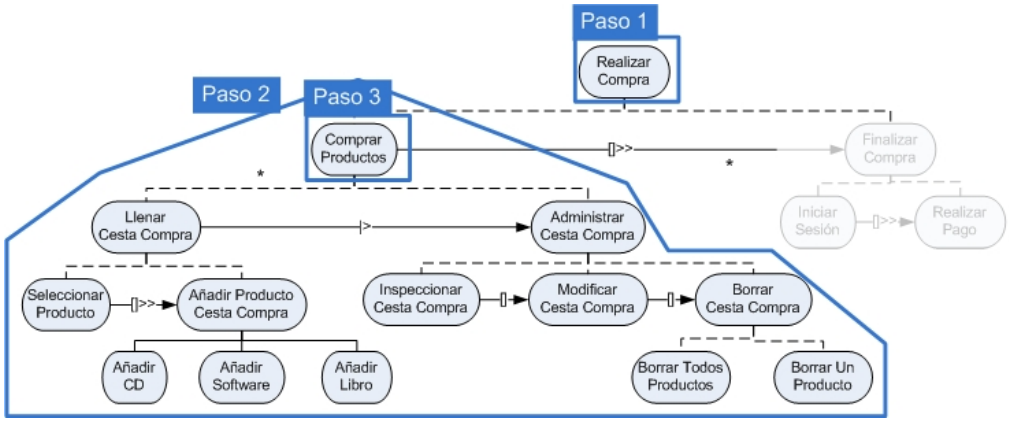

Figura A.1: Pasos 1, 2 y 3 del recorrido del modelo de tareas

Paso 1. Se comienza el recorrido por la raíz del modelo, cuyo nodo se denomina `Realizar Compra` (ver el paso 1 de la figura A.1).

3: No participa en una relación estructural

7: No es una tarea hoja

12: Recorre su subárbol izquierdo

Paso 2. Se selecciona el subárbol izquierdo de la tarea `Realizar Compra` (ver el paso 2 de la figura A.1).

Paso 3. La siguiente tarea a analizar es `Comprar Productos` (ver el paso 3 de la figura A.1).

3: No participa en una relación estructural, por lo tanto devuelve falso

7: No es una tarea hoja, por lo tanto devuelve falso

12: Recorre su subárbol izquierdo

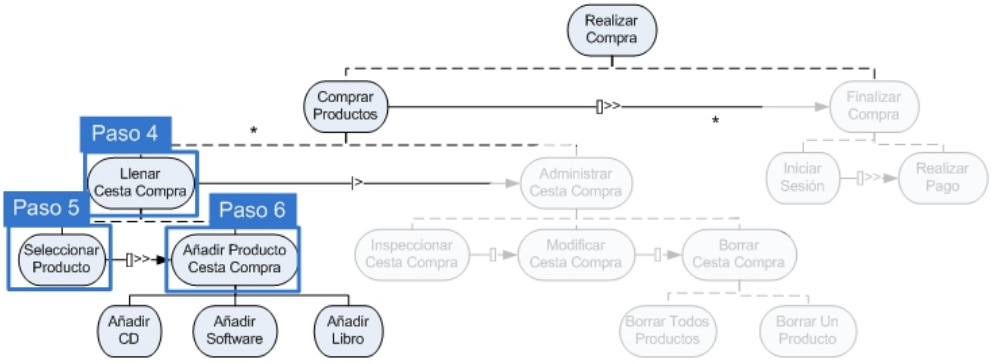

Figura A.2: Pasos 4, 5 y 6 del recorrido del modelo de tareas

Paso 4. A continuación, se analiza la tarea `Llenar Cesta Compra` (ver el paso 4 de la figura A.2).

3: No participa en una relación estructural, por lo tanto devuelve falso

7: No es una tarea hoja, por lo tanto devuelve falso

12: Recorre su subárbol izquierdo

Paso 5. La siguiente tarea seleccionada es `Seleccionar Producto` (ver el paso 5 de la figura A.2).

3: No participa en una relación estructural, por lo tanto devuelve falso

7: Es una tarea hoja pero es la primera de una relación temporal, por lo tanto devuelve falso

12: Finaliza el recorrido de este subárbol y vuelve a la tarea `Llenar Cesta Compra` (Paso 4)

Paso 6. A continuación se pasa a analizar en el subárbol derecho la tarea `Llenar Cesta Compra`, analizando la tarea `Añadir Producto Cesta Compra` (ver el paso 6 de la figura A.2).

3: No participa en una relación estructural, por lo tanto devuelve falso

7: No es una tarea hoja, por lo tanto devuelve falso

12: Recorre su subárbol izquierdo

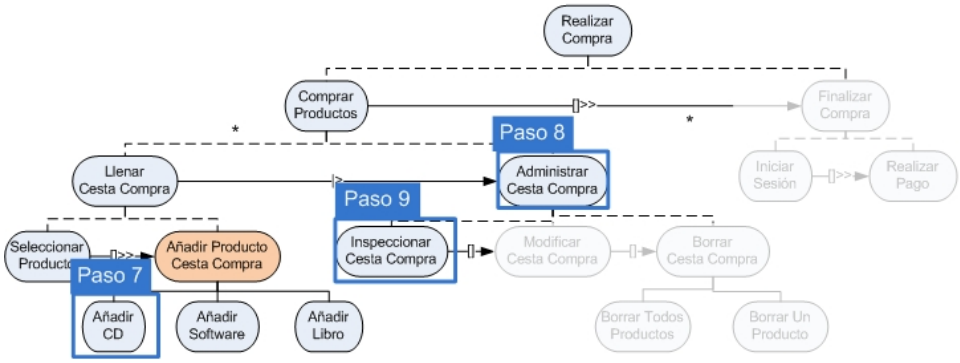

Figura A.3: Pasos 7, 8 y 9 del recorrido del modelo de tareas

Paso 7. La siguiente tarea a analizar es `Añadir CD` (ver el paso 7 de la figura A.3).

3: Participa en una relación estructural y es una tarea hoja, por lo tanto devuelve verdadero

4: Se selecciona la tarea superior `Añadir Producto Cesta Compra` como operación a ser publicada en el servicio Web

12: Finaliza el recorrido de este subárbol y vuelve a la tarea `Añadir Producto Cesta Compra`. También se finaliza el recorrido de esta tarea y por lo tanto se vuelve a la tarea `Comprar Productos` (Paso 2)

Paso 8. A continuación se pasa a la tarea `Administrar Cesta Compra` (ver el paso 9 de la figura A.3).

3: No participa en una relación estructural, por lo tanto devuelve falso

7: No es una tarea hoja, por lo tanto devuelve falso

12: Recorre su subárbol izquierdo

Paso 9. La siguiente tarea a analizar es la tarea `Inspeccionar Cesta Compra` (ver el paso 9 de la figura A.3).

3: No participa en una relación estructural, por lo tanto devuelve falso

7: Es una tarea hoja pero es la primera de una relación temporal, por lo tanto devuelve falso

12: Finaliza el recorrido de este subárbol y vuelve a la tarea `Administrar Cesta Compra` (Paso 8)

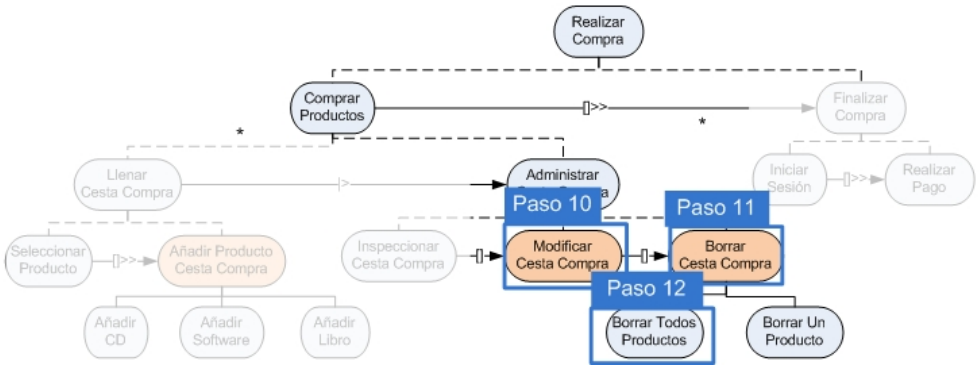

Figura A.4: Pasos 10, 11 y 12 del recorrido del modelo de tareas

Paso 10. En el siguiente paso se analiza la tarea `Modificar Cesta Compra` (ver el paso 10 de la figura A.4).

3: No participa en una relación estructural, por lo tanto devuelve falso

7: Es una tarea hoja y no es la primera de una relación temporal, por lo tanto devuelve verdadero

8: Se selecciona la tarea `Modificar Cesta Compra` como operación a ser publicada en el servicio Web

12: Finaliza el recorrido de este subárbol y vuelve a la tarea `Administrar Cesta Compra` (Paso 8)

Paso 11. A continuación se analiza la tarea `Borrar Cesta Compra` (ver el paso 11 de la figura A.4).

3: No participa en una relación estructural, por lo tanto devuelve falso

7: No es una tarea hoja, por lo tanto devuelve falso

12: Recorre su subárbol izquierdo

Paso 12. La tarea que se analiza en este paso es la tarea `Borrar Todos Productos` (ver el paso 12 de la figura A.4).

3: Participa en una relación estructural y es una tarea hoja, por lo tanto devuelve verdadero

7: Se selecciona la tarea superior `Borrar Cesta Compra` como operación a ser publicada en el servicio Web

12: Finaliza el recorrido de este subárbol y vuelve a la tarea `Borrar Cesta Compra`. También se finaliza el recorrido de esta tarea y vuelve a las tareas `Administrar Cesta Compra` y `Comprar Productos`, las cuales también acaban el recorrido y finalmente se vuelve a la tarea raíz `Realizar Compra` (Paso 1)

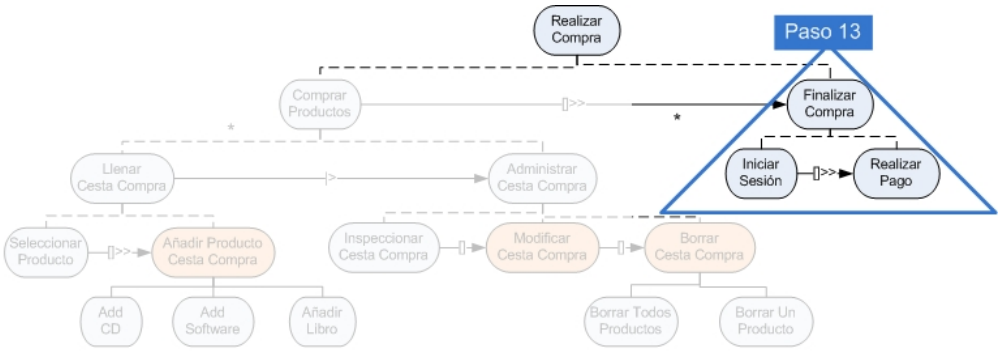

Figura A.5: Paso 13 del recorrido del modelo de tareas

Paso 13. Una vez finalizado el subárbol izquierdo de la tarea `Realizar Compra` a continuación se pasa a analizar su subárbol derecho (ver el paso 13 de la figura A.5).

Paso 14. Se selecciona la tarea `Finalizar Compra` para ser analizada (ver el paso 14 de la figura A.6).

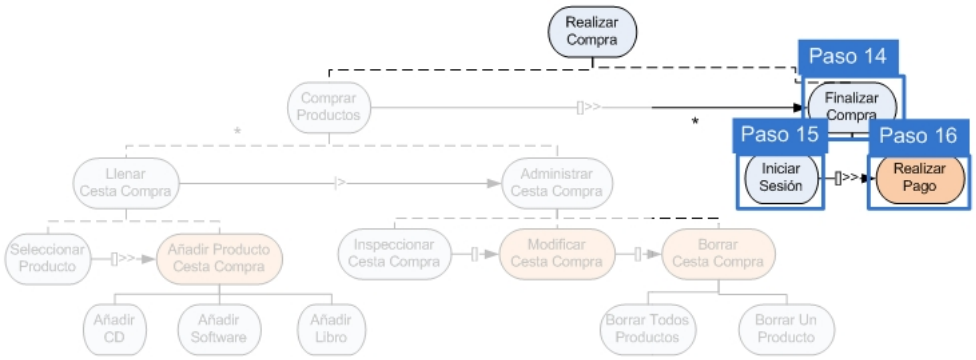

Figura A.6: Pasos 14, 15 y 16 del recorrido del modelo de tareas

3: No participa en una relación estructural, por lo tanto devuelve falso

7: No es una tarea hoja, por lo tanto devuelve falso

12: Recorre su subárbol izquierdo

Paso 15. La siguiente tarea a ser analizada es `Iniciar Sesión`(ver el paso 15 de la figura A.6).

3: No participa en una relación estructural, por lo tanto devuelve falso

7: Es una tarea hoja pero es la primera de una relación temporal, por lo tanto devuelve falso

12: Finaliza el recorrido de este subárbol y vuelve a la tarea `Finalizar Compra` (Paso 14)

Paso 16. Se analiza el último subárbol del modelo con la tarea `Realizar Pago` (ver el paso 16 de la figura A.6).

3: No participa en una relación estructural, por lo tanto devuelve falso

7: Es una tarea hoja y no es la primera de una relación temporal, por lo tanto devuelve verdadero

8: Se selecciona la tarea `Realizar Pago` como operación a ser publicada en el servicio Web

12: Finaliza el recorrido de este subárbol y vuelve a la tarea `Finalizar Compra`. También se finaliza el recorrido de

esta tarea y finalmente se vuelve a la tarea raíz `Realizar Compra` y finaliza la ejecución del algoritmo.

27: Finaliza la ejecución y devuelve el conjunto de operaciones identificadas

En la figura A.7 se muestran las operaciones y las tareas desde las que han sido identificadas tras realizar la traza del algoritmo.

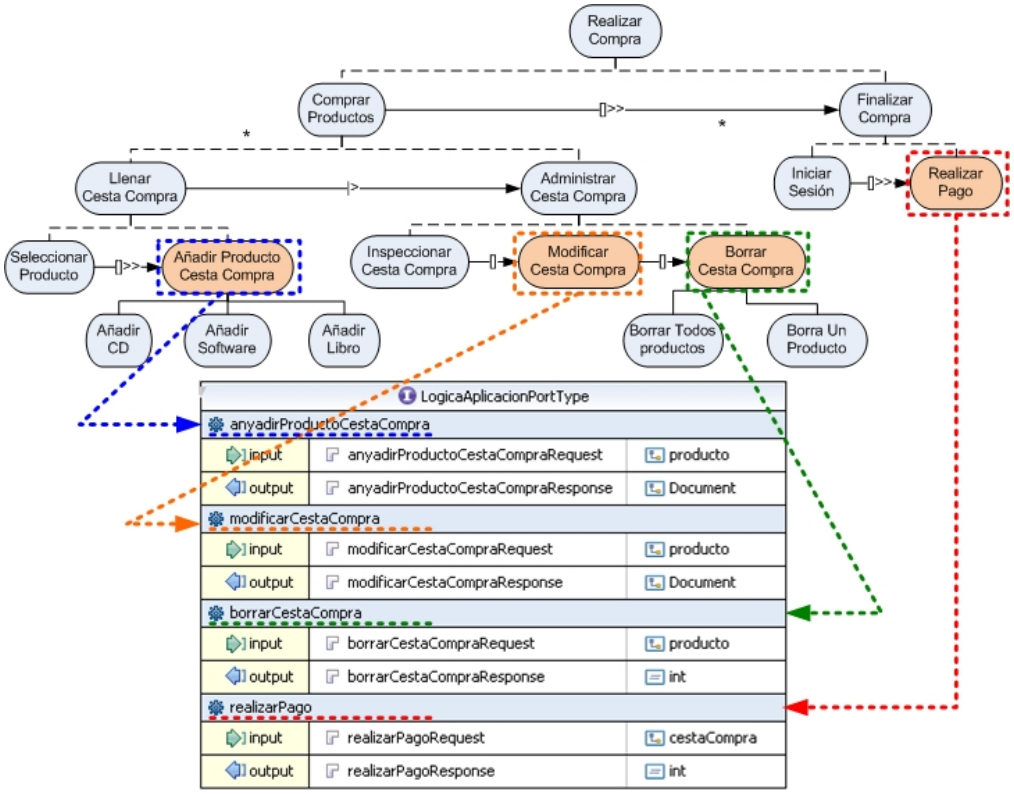

Figura A.7: Resumen de la identificación de operaciones para la Lógica de la Aplicación

Código de las operaciones

A continuación, para cada una de las operaciones identificadas, se detalla el código Java que las implementa (siguiendo los algoritmos 12, 13 y 14).

- anyadirProductoCestaCompra

```
public Document anaydirProductoCestaCompra(Producto
   pp_productoAnyadir){
  // Permite añadir un producto a la cesta de la compra. Los
     productos pueden ser CD, Software o Libro.
  try{
    frameworkWIF.executeService("Producto.MV_anyadirProducto",
       pp_productoAnyadir.toString());
  } catch(Exception e) { e.getMessage(); }
return frameworkWIF.getResponse(); }
```

- modificarCestaCompra

```
public Document modificarCestaCompra(Producto
   pp_productoAModificar){
  //Permite modificar un producto que se encuentra en la cesta
     de la compra.
  try{
    frameworkWIF.executeService("Producto.
       MV_modificarCestaCompra", pp_productoAModificar.toString
       ());
  } catch(Exception e) { e.getMessage(); }
  return frameworkWIF.getResponse(); }
```

- borrarCestaCompra

```
public int borrarCestaCompra(Producto pp_productoABorrar){
  //Permite eliminar un producto o todos los productos que se
     encuentran en la cesta.
  Boolean lb_token = Boolean.FALSE;
  try{
    lb_token = frameworkWIF.executeService("Producto.
       MV_borrarCestaCompra", pp_productoABorrar.toString());
  } catch(Exception e) { e.getMessage(); }
  if (lb_token) {return 0;} else {return -1;} }
```

- `RealizarPago`

```
public int realizarPago(CestaCompra pcc_CestaCompra){
  //Permite administrar el pago de las compras realizadas.
  Boolean lb_token = Boolean.FALSE;
  try{
    lb_token = frameworkWIF.executeService("CestaCompra.
       MV_realizarPago", pcc_CestaCompra.toString());
  } catch(Exception e) { e.getMessage(); }
  if (lb_token) {return 0;} else {return -1;} }
```

A.2.2 Gestión de Usuarios

A continuación, se describe la aplicación del algoritmo propuesto (ver el algoritmo 3 en la página 109) sobre el caso de estudio. Este algoritmo se aplica para identificar, a partir del modelo de usuarios, las operaciones a publicar en el servicio Web. Para cada uno de los pasos descritos se enumera la línea del algoritmo.

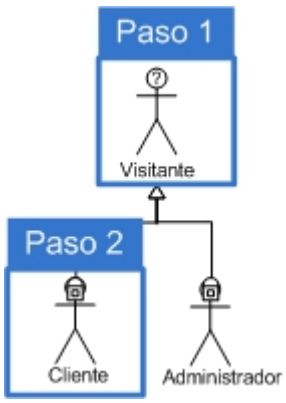

Figura A.8: Pasos 1 y 2 del recorrido del modelo de usuarios

Paso 1. Se comienza el recorrido por la raíz del modelo (ver el paso 1 de la figura A.8). La raíz corresponde con el tipo de usuario `Visitante`.

1: No es un usuario que necesita identificación, por lo tanto devuelve falso

20: Recorre su subárbol izquierdo

Paso 2. El siguiente tipo de usuario a analizar es `Cliente` (ver el paso 2 de la figura A.8).

1: Es un usuario que necesita identificación, por lo tanto devuelve verdadero

2-18: Se seleccionan las operaciones de gestión de usuarios para ser publicadas en el servicio Web

24: Finaliza la ejecución y devuelve el conjunto de operaciones identificadas

En la figura A.9 se muestran las operaciones y los usuarios desde las que han sido identificadas tras realizar la traza del algoritmo.

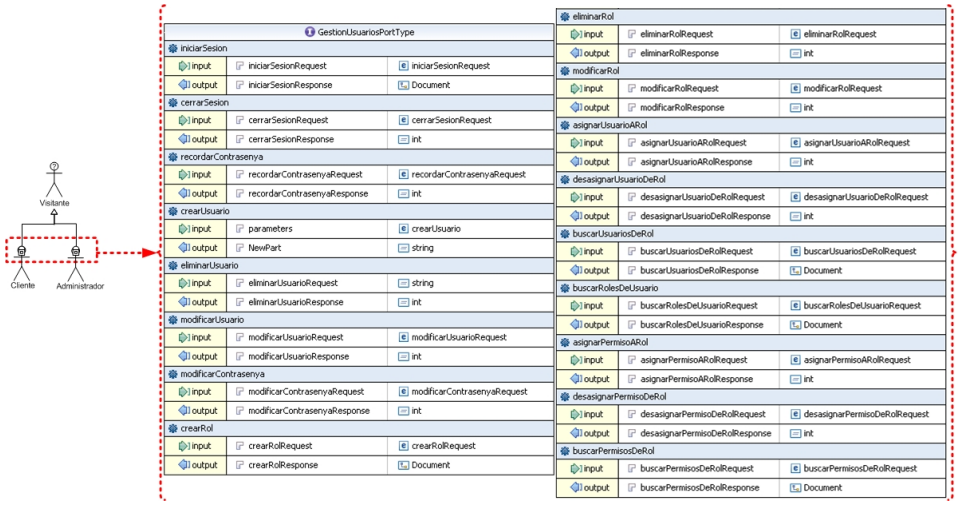

Figura A.9: Resumen de la identificación de operaciones para la Gestión de Usuarios

Código de las operaciones

A continuación, para cada una de las operaciones identificadas, se detalla el código Java que las implementa:

- iniciarSesion

```
public Document iniciarSesion(String ps_Rol, String ps_Usuario,
    String ps_Contrasenya){
  //Permite iniciar la sesión de un usuario.
  try{
    frameworkWIF.executeService(ps_Rol + ".MV_AgentValidation",
        ps_Usuario+","+ps_Contrasenya);
  } catch(Exception e) { e.getMessage(); }
  return frameworkWIF.getResponse(); }
```

- cerrarSesion

```
public int cerrarSesion(String ps_Sesion){
  //Permite cerrar la sesión de un usuario.
  Boolean lb_token = Boolean.FALSE;
  try{
    lb_token = frameworkWIF.executeService("Sesion.
       MV_CloseSesion", ps_Sesion);
  } catch(Exception e) { e.getMessage(); }
  if (lb_token) {return 0;} else {return -1;} }
```

- recordarContrasenya

```
public int recordarContrasenya(String ps_Usuario, String
   ps_Respuesta){
//Envia un correo al usuario para recordarle su contrasenya si
   contesta correctamente a la pregunta secreta.
  Boolean lb_token = Boolean.FALSE;
  try{
    lb_token = frameworkWIF.executeService("User.
       MV_RemindPassword", ps_Usuario+","+ps_Respuesta);
  }catch(Exception e)   { e.getMessage(); }
  if (lb_token) {return 0;} else {return -1;} }
```

- crearUsuario

```
public int crearUsuario(String ps_Rol, Usuario pu_DatosUsuario)
   {
  //Crea un nuevo usuario y le asigna un rol.
  Boolean lb_token = Boolean.FALSE;
  try{
    lb_token = frameworkWIF.executeService("User.
       create_instance", ps_Rol+","+pu_DatosUsuario.toString())
       ;
  }catch(Exception e)   { e.getMessage(); }
  if (lb_token) {return 0;} else {return -1;} }
```

- eliminarUsuario

```
public int eliminarUsuario(String ps_UsuarioID){
  //Elimina un usuario existente.
  Boolean lb_token = Boolean.FALSE;
  try{
    lb_token = frameworkWIF.executeService("User.
       delete_instance",ps_UsuarioID);
  }catch(Exception e)   { e.getMessage();}
  if (lb_token) {return 0;} else {return -1;} }
```

- modificarUsuario

```
public int modificarUsuario(Usuario pu_DatosUsuario){
  //Modifica los datos de un usuario existente.
  Boolean lb_token = Boolean.FALSE;
  try{
    lb_token = frameworkWIF.executeService("User.edit_instance"
       , pu_DatosUsuario.toString());
  }catch(Exception e)   { e.getMessage(); }
  if (lb_token) {return 0;} else {return -1;} }
```

- modificarContrasenya

```
public int modificarContrasenya(String ps_ContrasenyaAnt,
   String ps_ContrasenyaNue){
  //Modifica la contrasenya de un usuario.
  Boolean lb_token = Boolean.FALSE;
  try{
    lb_token = frameworkWIF.executeService("User.
       MV_ChangePassWord", ps_ContrasenyaAnt+","+
       ps_ContrasenyaNue);
  }catch(Exception e)   { e.getMessage(); }
  if (lb_token) {return 0;} else {return -1;} }
```

- asignarUsuarioARol

```
public int asignarUsuarioARol(String ps_Rol, String
   ps_UsuarioID){
  //Asigna un usuario a un rol.
  Boolean lb_token = Boolean.FALSE;
  try{
    lb_token = frameworkWIF.executeService(ps_Rol+".MV_AddUser"
       , ps_UsuarioID);
  }catch(Exception e)   { e.getMessage(); }
  if (lb_token) {return 0;} else {return -1;} }
```

- desasignarUsuarioDeRol

```
public int desasignarUsuarioDeRol(String ps_Rol, String
   ps_UsuarioID){
  //Desasigna un usuario a un rol.
  Boolean lb_token = Boolean.FALSE;
  try{
    lb_token = frameworkWIF.executeService(ps_Rol+".MV_DelUser"
       , ps_UsuarioID);
  }catch(Exception e)   { e.getMessage(); }
  if (lb_token) {return 0;} else {return -1;} }
```

- buscarUsuariosDeRol

```
public Document buscarUsuariosDeRol(String ps_Rol, String
   ps_RolID){
  //Busca los usuarios de un rol.
  return frameworkWIF.queryRelated(ps_Rol, "nombre", ps_RolID,"
     UsuariosAsignados"); }
```

- buscarRolesDeUsuario

```
public Document buscarRolesDeUsuario(Usuario pu_Usuario, String
    ps_UsuarioID){
  //Busca los usuarios de un rol.
  return frameworkWIF.queryRelated(pu_Usuario.toString(), "
     nombre", ps_UsuarioID, "UsuariosAsignados", "");  }
```

- asignarPermisosARol

```
public int asignarPermisosARol(String ps_Rol, String
   ps_PermisoID){
  //Asigna un usuario a un rol.
  Boolean lb_token = Boolean.FALSE;
  try{
    lb_token = frameworkWIF.executeService(ps_Rol+".
       MV_AddPermission", ps_PermisoID);
  }catch(Exception e)   { e.getMessage(); }
  if (lb_token) {return 0;} else {return -1;}   }
```

- desasignarPermisosDeRol

```
public int desasignarPermisosDeRol(String ps_Rol, String
   ps_PermisoID){
  //Desasigna un usuario a un rol.
  Boolean lb_token = Boolean.FALSE;
  try{
    lb_token = frameworkWIF.executeService(ps_Rol+".
       MV_DelPermission", ps_PermisoID);
  }catch(Exception e)   { e.getMessage(); }
  if (lb_token) {return 0;} else {return -1;} }
```

- `buscarPermisosDeRol`

```
public Document buscarPermisosDeRol(String ps_Rol, String
    ps_RolID){
  //Busca los usuarios de un rol.
  return frameworkWIF.queryRelated(ps_Rol, "nombre", ps_RolID,
     "PermisosAsignados", ""); }
```

A.2.3 Recuperación de Información

A continuación, se describe la aplicación de los algoritmos propuestos (ver los algoritmos: 4 en la página 119, 5 en la página 120, 6 en la página 120, 7 en la página 121) sobre el caso de estudio. Este algoritmo se aplica para identificar, a partir del contexto navegacional, las operaciones a publicar en el servicio Web. Para cada uno de los pasos descritos se enumera la línea del algoritmo.

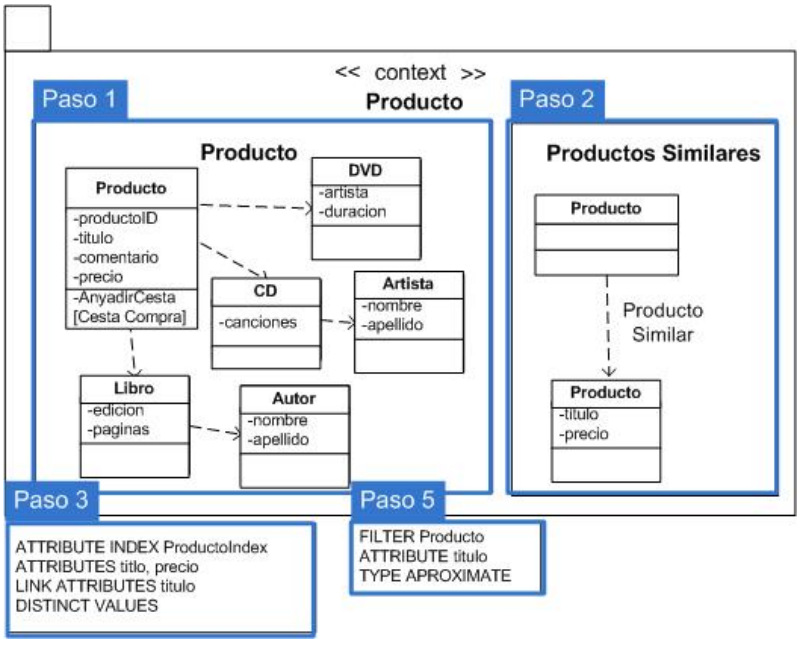

Figura A.10: Pasos 1, 2, 3 y 5 del recorrido del contexto `Producto`

Paso 1. Recorrer las UAIs del contexto. La primera UAI a ser analizada es `Producto` (ver el paso 1 de la figura A.10).

1: Recorre la primera UAI

2: Se selecciona la operación `recuperarProducto` para ser publicada en el servicio Web

4: Continúa recorriendo las UAI del contexto

Paso 2. La siguiente UAI a ser analizada es `Productos Similares` (ver el paso 2 de la figura A.10).

1: Recorre la segunda UAI

2: Se selecciona la operación `recuperarProductosSimilares` para ser publicada en el servicio Web

4: No existen más UAI en el contexto

Paso 3. Se recorren los índices. El primer índice a ser analizado es `ProductoIndex` (ver el paso 3 en la figura A.10).

5: Recorre el primer índice

6: Se selecciona la operación `indexarProductoIndex` para ser publicada en el servicio Web

8: No existen más índices en el contexto

Paso 4. Se recorren las condiciones de población.

9: Recorre las condiciones de población, no encuentra ninguna

Paso 5. Se recorren los filtros. El primer filtro a ser analizado es `Producto` (ver el paso 5 en la figura A.10).

13: Recorre el primer filtro

14: Se selecciona la operación `buscarProducto` para ser publicada en el servicio Web

16: No existen más filtros en el contexto

17: Finaliza la ejecución y devuelve el conjunto de operaciones identificadas

En la figura A.11 se muestran las operaciones y de qué elementos del contexto han sido identificadas tras realizar la traza del algoritmo.

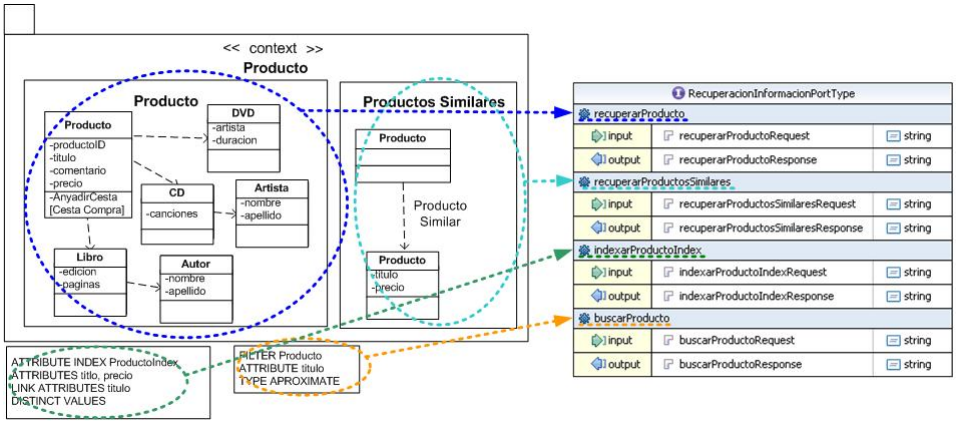

Figura A.11: Resumen de la identificación de operaciones para la Recuperación de Información

Código de las operaciones

A continuación, para cada una de las operaciones identificadas, se detalla el código Java que las implementa.

- `recuperarProducto`

```
public Document recuperarProducto(String ps_productoId){
//Permite recuperar la información todos los productos (si no
   se le pasa ningún parámetro) o de un producto en particular
   (si se pasa su identificador como parámetro.
Document ld_recuperarProducto = null;
Document ld_Producto = null;
try {
//PASO1
if (ps_productoId.isEmpty()){ //Recupera toda la población
 ld_Producto.appendChild(frameworkWIF.queryPopulation("Producto
    ", "productoId,titulo,comentario,precio"));
}
else{ //Recupera una instancia en concreto
 ld_Producto.appendChild(frameworkWIF.queryById("Producto", "
    titulo,comentario,precio", ps_productoId));
}
ld_recuperarProducto.appendChild(ld_Producto);
```

```
//PASO 2
Producto lp_ProductoAProcesar = (Producto) ld_Producto.
   getFirstChild();
Document ld_DvdRelacionados = null;
Document ld_CdRelacionados = null;
Document ld_LibroRelacionados = null;
while (lp_ProductoAProcesar != null){
 String ls_ProductoID = lp_ProductoAProcesar.getProductoId();
 //DVD
 ld_DvdRelacionados.appendChild(frameworkWIF.queryRelated("
    Producto", "artista,duracion",  ls_ProductoID, "Ddv"));
 //CD
 ld_CdRelacionados.appendChild(frameworkWIF.queryRelated("
    Producto", "canciones",  ls_ProductoID, "Cd"));
 //Libro
 ld_LibroRelacionados.appendChild(frameworkWIF.queryRelated("
    Producto", "edicion,paginas",  ls_ProductoID, "Libro"));
 lp_ProductoAProcesar = (Producto) ld_Producto.getElementById(
    ls_ProductoID).getNextSibling();
}
ld_recuperarProducto.appendChild(ld_DvdRelacionados);
ld_recuperarProducto.appendChild(ld_CdRelacionados);
ld_recuperarProducto.appendChild(ld_LibroRelacionados);

//PASO 3
Cd lcd_CdAProcesar = (Cd) ld_CdRelacionados.getFirstChild();
Document ld_ArtistasRelacionados = null;
while (lcd_CdAProcesar != null){
 String ls_CdId = lcd_CdAProcesar.getCdId();
 ld_ArtistasRelacionados.appendChild(frameworkWIF.queryRelated(
    "Cd", "nobmre,apellido", ls_CdId, "Artista"));
 lcd_CdAProcesar = (Cd) ld_CdRelacionados.getElementById(
    ls_CdId).getNextSibling();
}
ld_recuperarProducto.appendChild(ld_ArtistasRelacionados);
//Autores_Libros
Libro ll_LibroAProcesar = (Libro) ld_LibroRelacionados.
   getFirstChild();
Document ld_AutoresRelacionados = null;
while (ll_LibroAProcesar != null){
 String ls_LibroId = ll_LibroAProcesar.getLibroId();
 ld_AutoresRelacionados.appendChild(frameworkWIF.queryRelated("
    Libro", "nombre,apellido",  ls_LibroId, "Autor"));
}
ld_recuperarProducto.appendChild(ld_AutoresRelacionados);
```

```
} catch(Exception e)    { e.getMessage(); }
return cp_Producto; }
```

- recuperarProductosSimilares

```
public Document recuperarProductosSimilares(String
   ps_productoId){
//Esta operación permite recuperar la información de todos los
   productos similares a otro producto o conjunto de productos
Document ld_recuperarProductosSimilares = null;
Document ld_ProductoSimilar = null;
try{
  //PASO 1
  if (ps_productoId.isEmpty()){
   ld_ProductoSimilar.appendChild(frameworkWIF.queryPopulation(
      "Producto", "productoID,titulo,comentario,precio"));
 }
 else{
 ld_ProductoSimilar.appendChild(frameworkWIF.queryById("
    Producto", "titulo,comentario,precio", ps_productoId));
 }
 ld_recuperarProductosSimilares.appendChild(ld_ProductoSimilar)
    ;

  //PASO 2
  Producto lp_ProductoSimilarAProcesar = (Producto)
     ld_ProductoSimilar.getFirstChild();
  Document ld_ProductosSimilaresRelacionados = null;
  while (lp_ProductoSimilarAProcesar != null){
    String ls_ProductoSimilarId = lp_ProductoSimilarAProcesar.
       getProductoId();
    ld_ProductosSimilaresRelacionados.appendChild(frameworkWIF.
       queryRelated("Producto", "titulo,comentario,precio",
       ls_ProductoSimilarId, "ProductosSimilares"));
   lp_ProductoSimilarAProcesar = (Producto) ld_ProductoSimilar.
      getElementById(ls_ProductoSimilarId).getNextSibling();
  }
  ld_recuperarProductosSimilares.appendChild(
     ld_ProductosSimilaresRelacionados);
}catch(Exception e)     { e.getMessage(); }
return ld_recuperarProductosSimilares; }
```

- recuperarCategoriasProductos

```
public Document recuperarCategoriasProductos(String
   ps_categoriaProductoId){
//Esta operación permite recuperar las categorías de los
   productos existentes o la categoría de un producto en
   particular.
 Document ld_recuperarCategoriasProductos = null;
 Document ld_CategoriasProductos = null;
 try{
  //PASO 1
  if (ps_categoriaProductoId.isEmpty()){
   ld_CategoriasProductos.appendChild(frameworkWIF.
      queryPopulation("CategoriaProducto", "nombre"));
  } else{
   ld_CategoriasProductos.appendChild(frameworkWIF.queryById("
      CategoriaProducto", "nombre", ps_categoriaProductoId));
  }
  ld_recuperarCategoriasProductos.appendChild(
     ld_CategoriasProductos);
  //PASO 2
  CategoriaProducto lp_CategoriaProductoAProcesar = (
     CategoriaProducto) ld_CategoriasProductos.getFirstChild();
  Document ld_ProductosRelacionados = null;
  while (lp_CategoriaProductoAProcesar != null){
   String ls_CategoriaProductoId =
      lp_CategoriaProductoAProcesar.getCategoriaProductoId();
   ld_ProductosRelacionados.appendChild(frameworkWIF.
      queryRelated("CategoriaProducto", "titulo,comentario,
      precio",  ls_CategoriaProductoId, "CategoriaProducto"));
   lp_CategoriaProductoAProcesar = (CategoriaProducto)
      ld_CategoriasProductos.getElementById(
      ls_CategoriaProductoId).getNextSibling();
  }
  ld_recuperarCategoriasProductos.appendChild(
     ld_ProductosRelacionados);
}catch(Exception e)     { e.getMessage(); }
  return ld_recuperarCategoriasProductos; }
```

- recuperarComentariosClientes

```
public Document recuperarComentariosClientes(String
   ps_comentariosClientesId){
```

```
//Esta operación permite recuperar los comentarios de los
   clientes sobre ciertos productos o sobre un producto en
   particular.
Document ld_recuperarComentariosClientes = null;
Document ld_ComentariosClientes = null;
try{
 //PASO 1
 if (ps_comentariosClientesId.isEmpty()){
  ld_ComentariosClientes.appendChild(frameworkWIF.
     queryPopulation("ComentarioCliente", "comentarioClienteId
     ,comentario"));
 } else{
  ld_ComentariosClientes.appendChild(frameworkWIF.queryById("
     ComentarioCliente", "comentarioClienteId,comentario",
     ps_comentariosClientesId));
 }
 ld_recuperarComentariosClientes.appendChild(
    ld_ComentariosClientes);
 //PASO 2
 ComentarioCliente lcc_ComentarioClienteAProcesar = (
    ComentarioCliente) ld_ComentariosClientes.getFirstChild();
 Document ld_ClientesRelacionados = null;
 Document ld_ProductosRelacionados = null;
 while (lcc_ComentarioClienteAProcesar != null){
  String ls_ComentarioClienteId =
     lcc_ComentarioClienteAProcesar.getComentarioClienteId();
  ld_ClientesRelacionados.appendChild(frameworkWIF.
     queryRelated("ComentarioCliente", "nombre,apellidos,
     ciudad,pais",  ls_ComentarioClienteId, "Cliente"));
  ld_ProductosRelacionados.appendChild(frameworkWIF.
     queryRelated("ComentarioCliente", "titulo",
     ls_ComentarioClienteId, "ProductoComentado"));
  lcc_ComentarioClienteAProcesar = (ComentarioCliente)
     ld_ComentariosClientes.getElementById(
     ls_ComentarioClienteId).getNextSibling();
 }
ld_recuperarComentariosClientes.appendChild(
   ld_ClientesRelacionados);
ld_recuperarComentariosClientes.appendChild(
   ld_ProductosRelacionados);
}catch(Exception e)     { e.getMessage(); }
 return ld_recuperarComentariosClientes; }
```

- recuperarClientes

```
public Document recuperarClientes(String ps_clienteId){
//Esta operación permite recuperar los datos de todos los
   clientes o de un cliente en particular
Document ld_recuperarClientes = null;
Document ld_Clientes = null;
try{
 //PASO 1
 if (ps_clienteId.isEmpty()){
  ld_Clientes.appendChild(frameworkWIF.queryPopulation("Cliente
     ", "nombre,apellidos,ciudad,pais"));
 } else{
  ld_Clientes.appendChild(frameworkWIF.queryById("Cliente", "
     nombre,apellidos,ciudad,pais", ps_clienteId));
 }
 ld_recuperarClientes.appendChild(ld_Clientes);
 //PASO 2
 Cliente lp_ClienteAProcesar = (Cliente) ld_Clientes.
    getFirstChild();
 Document ld_ComentariosClienteRelacionados = null;
 while (lp_ClienteAProcesar != null){
  String ls_ClienteId = lp_ClienteAProcesar.getClienteId();
  ld_ComentariosClienteRelacionados.appendChild(frameworkWIF.
     queryRelated("Cliente", "comentario",  ls_ClienteId, "
     ComentarioCliente"));
  lp_ClienteAProcesar = (Cliente)
  ld_Clientes.getElementById(ls_ClienteId).getNextSibling();
 }
 ld_recuperarClientes.appendChild(
    ld_ComentariosClienteRelacionados);
 //PASO 3
 //Artistas_CD
 ComentarioCliente lcc_ComentarioClienteAProcesar = (
    ComentarioCliente) ld_ComentariosClienteRelacionados.
    getFirstChild();
 Document ld_ProductosRelacionados = null;
 while (lcc_ComentarioClienteAProcesar != null){
  String ls_ComentarioClienteId =
     lcc_ComentarioClienteAProcesar.getComentarioClienteId();
  ld_ProductosRelacionados.appendChild(frameworkWIF.
     queryRelated("ComentarioCliente", "titulo",
     ls_ComentarioClienteId, "ProductoComentado"));
  lcc_ComentarioClienteAProcesar = (ComentarioCliente)
     ld_ComentariosClienteRelacionados.getElementById(
```

```
     ls_ComentarioClienteId).getNextSibling();
 }
 ld_recuperarClientes.appendChild(ld_ProductosRelacionados);
}catch(Exception e)     { e.getMessage(); }
 return ld_recuperarClientes; }
```

- recuperarVendedores

```
public Document recuperarVendedores(String ps_vendedorId){
//Esta operación permite recuperar los datos de todos los
   vendedores o de un vendedor en particular.
Document ld_recuperarVendedores = null;
Document ld_Vendedores = null;
try{
 //PASO 1
 if (ps_vendedorId.isEmpty()){
  ld_Vendedores.appendChild(frameworkWIF.queryPopulation("
     Vendedor", "nombre,ciudad,pais"));
 } else{
  ld_Vendedores.appendChild(frameworkWIF.queryById("Vendedor",
     "nombre,ciudad,pais", ps_vendedorId));
 }
 ld_recuperarVendedores.appendChild(ld_Vendedores);
 //PASO 2
 Vendedor lp_VendedorAProcesar = (Vendedor) ld_Vendedores.
    getFirstChild();
 Document ld_ProductosRelacionados = null;
 while (lp_VendedorAProcesar != null){
  String ls_VendedorId = lp_VendedorAProcesar.getVendedorId();
  ld_ProductosRelacionados.appendChild(frameworkWIF.
     queryRelated("Vendedor", "titulo",  ls_VendedorId, "
     Productos"));
  lp_VendedorAProcesar = (Vendedor) ld_Vendedores.
     getElementById(ls_VendedorId).getNextSibling();
 }
 ld_recuperarVendedores.appendChild(ld_ProductosRelacionados);
}catch(Exception e)     { e.getMessage(); }
 return ld_recuperarVendedores; }
```

- indexarProductoIndex

```
public Document indexarProductoIndex(){
//Esta operación provee un acceso indexado a la población de
   productos por su nombre y precio.
Document ld_indexarProductoIndex = null;
try {
 //PASO1
 //Recupera toda la población para el índice definido
 ld_indexarProductoIndex.appendChild(frameworkWIF.
    queryPopulation("Producto", "titulo,precio"));
} catch(Exception e) { e.getMessage(); }
 return ld_indexarProductoIndex; }
```

- buscarProducto

```
public Document buscarProducto(String ps_Filtro){
//Esta operación filtra el espacio de productos que recupera
   por nombre.
Document ld_buscarProducto = null;
Document ld_Producto = null;
try {
 //PASO1
 //Recupera toda la población
 ld_Producto.appendChild(frameworkWIF.queryPopulation("Producto
    ", "productoId,titulo,comentario,precio", "titulo␣like" +
    ps_Filtro));
 ld_buscarProducto.appendChild(ld_Producto);

 //PASO 2
 Producto lp_ProductoAProcesar = (Producto) ld_Producto.
    getFirstChild();
 Document ld_DvdRelacionados = null;
 Document ld_CdRelacionados = null;
 Document ld_LibroRelacionados = null;
 while (lp_ProductoAProcesar != null){
  String ls_ProductoID = lp_ProductoAProcesar.getProductoId();
  //DVD
  ld_DvdRelacionados.appendChild(frameworkWIF.queryRelated("
     Producto", "artista,duracion",  ls_ProductoID, "Ddv"));
  //CD
  ld_CdRelacionados.appendChild(frameworkWIF.queryRelated("
     Producto", "canciones",  ls_ProductoID, "Cd"));
```

```
 //Libro
                        ld_LibroRelacionados.appendChild(
                           frameworkWIF.queryRelated("Producto"
                           , "edicion,paginas",  ls_ productoID,
                            "Libro"));
 lp_ProductoAProcesar = (Producto) ld_Producto.getElementById(
    ls_ProductoID).getNextSibling();
}
ld_buscarProducto.appendChild(ld_DvdRelacionados);
ld_buscarProducto.appendChild(ld_CdRelacionados);
ld_buscarProducto.appendChild(ld_LibroRelacionados);

//PASO 3
//Artistas_CD
Cd lcd_CdAProcesar = (Cd) ld_CdRelacionados.getFirstChild();
Document ld_ArtistasRelacionados = null;
while (lcd_CdAProcesar != null){
 String ls_CdId = lcd_CdAProcesar.getCdId();
 ld_ArtistasRelacionados.appendChild(frameworkWIF.queryRelated
    ("Cd", "nobmre,apellido", ls_CdId, "Artista"));
 lcd_CdAProcesar = (Cd) ld_CdRelacionados.getElementById(
    ls_CdId).getNextSibling();
}
ld_buscarProducto.appendChild(ld_ArtistasRelacionados);
//Autores_Libros
Libro ll_LibroAProcesar = (Libro) ld_LibroRelacionados.
   getFirstChild();
Document ld_AutoresRelacionados = null;
while (ll_LibroAProcesar != null){
 String ls_LibroId = ll_LibroAProcesar.getLibroId();
 ld_AutoresRelacionados.appendChild(frameworkWIF.queryRelated(
    "Libro", "nombre,apellido",  ls_LibroId, "Autor"));
 ll_LibroAProcesar = (Libro) ld_LibroRelacionados.
    getElementById(ls_LibroId).getNextSibling();
}
ld_buscarProducto.appendChild(ld_AutoresRelacionados);

} catch(Exception e) {  e.getMessage(); }
 return ld_buscarProducto; }
```

- buscarVendedores

```
public Document buscarVendedores(String ps_Filtro){
//Esta operación filtra el espacio de vendedores que recupera
   por nombre.
Document ld_buscarVendedores = null;
Document ld_Vendedores = null;
try{
 //PASO 1
 ld_Vendedores.appendChild(frameworkWIF.queryPopulation("
    Vendedor", "nombre,ciudad,pais","Vendedor␣=␣"+ps_Filtro));
 ld_buscarVendedores.appendChild(ld_Vendedores);
 //PASO 2
 Vendedor lp_VendedorAProcesar = (Vendedor) ld_Vendedores.
    getFirstChild();
 Document ld_ProductosRelacionados = null;
 while (lp_VendedorAProcesar != null){
  String ls_VendedorId = lp_VendedorAProcesar.getVendedorId();
  ld_ProductosRelacionados.appendChild(frameworkWIF.
     queryRelated("Vendedor", "titulo",  ls_VendedorId, "
     Productos"));
  lp_VendedorAProcesar = (Vendedor) ld_Vendedores.
     getElementById(ls_VendedorId).getNextSibling();
 }
 ld_buscarVendedores.appendChild(ld_ProductosRelacionados);
}catch(Exception e)     { e.getMessage(); }
 return ld_buscarVendedores; }
```

A.2.4 Soporte a la Navegación

A continuación, se describe la aplicación de los algoritmos propuestos (ver los algoritmos 8, 9 y 10 en las páginas 128, 128 y 129 respectivamente) sobre el caso de estudio. Este algoritmo identifica, a partir del mapa y del contexto navegacional, las operaciones a publicar en el servicio Web. Para cada uno de los pasos descritos se enumera la línea del algoritmo.

Paso 1. Se comienza el recorrido analizando si existen enlaces de Exploración en el mapa navegacional (ver el paso 1 en la figura A.12).

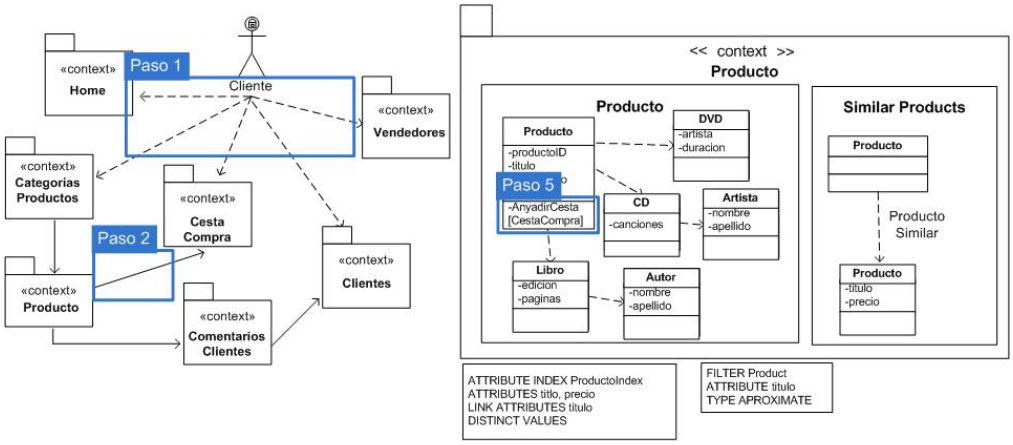

Figura A.12: Pasos 1, 2 y 5 del recorrido del mapa y del contexto navegacional

1: Existen enlaces de exploración en el mapa, por lo tanto devuelve verdadero

2: Se selecciona la operación `recuperarEnlacesExploracion` para ser publicada en el servicio Web

Paso 2. A continuación, se analiza si existen enlaces de Secuencia en el mapa navegacional (ver el paso 2 en la figura A.12).

5: Existen enlaces de secuencia en el mapa, por lo tanto devuelve verdadero

6: Se selecciona la operación `recuperarEnlacesSecuencia` para ser publicada en el servicio Web

Paso 3. Se analiza si existen enlaces de Servicio en algún contexto del mapa navegacional, para ello se recorren los contexto del mapa navegacional.

9: Se recorre el primer contexto

10: El contexto `Home` no tiene definido ningún enlace de servicio, por lo que continúa recorriendo los siguientes contextos

Paso 4. Se continúa analizando el segundo contexto del mapa, el contexto `Categorías Productos`.

9: Se recorre el segundo contexto

10: El contexto `Categorías Productos` no tiene definido ningún enlace de servicio, por lo que continúa recorriendo los siguientes contextos

Paso 5. Se continúa analizando el tercer contexto del mapa, el contexto `Producto` (ver el paso 5 en la figura A.12).

9: Se recorre el tercer contexto

10: El contexto `Producto` si que tiene definido un enlace de servicio, por lo que devuelve verdadero

11: Se selecciona la operación `recuperarEnlaceServicio` para ser publicada en el servicio Web

15: Finaliza la ejecución y devuelve el conjunto de operaciones identificadas

En la figura A.13 se muestran las operaciones y desde qué elementos, de los mapas navegacionales y contextos, han sido identificadas tras realizar la traza del algoritmo.

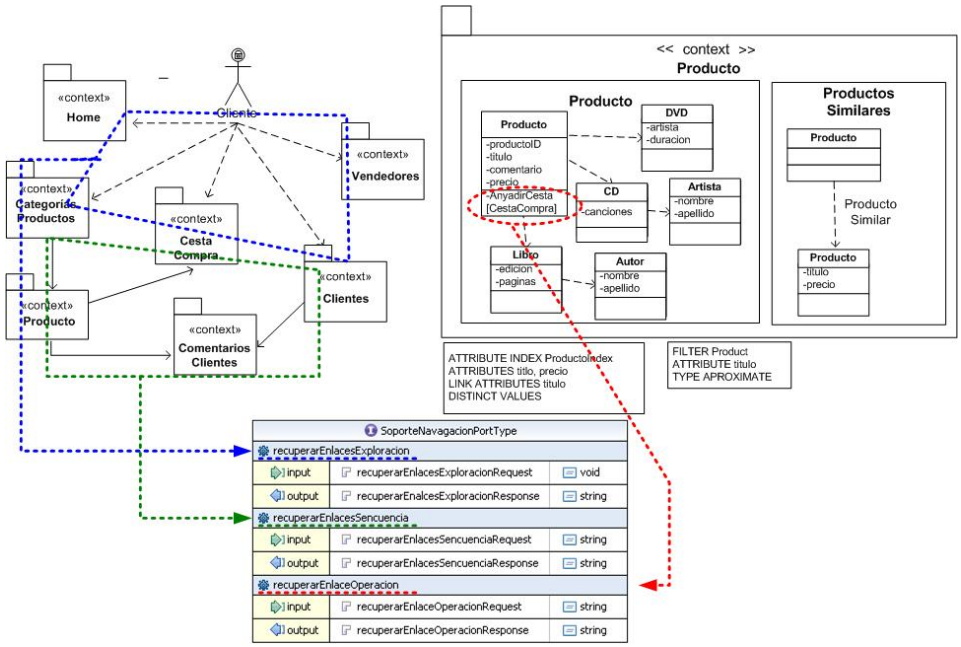

Figura A.13: Operaciones para el soporte a la navegación

Código de las operaciones

A continuación, para cada una de las operaciones identificadas , se detalla el código Java que las implementa. Cabe recordar que este grupo funcional no utiliza ninguna implementación generada por OlivaNOVA.

- `recuperarEnlacesExploracion`

```
public String recuperarEnlacesExploracion(){
  return "Home, Categorias Productos, Cesta Compra, Clientes,
     Vendedores";
}
```

- `recuperarEnlaceSecuencia`

```
public String recuperarEnlaceSencuencia(String ps_contexto){
  if (ps_contexto == "Categorias Productos") { return "producto
     "; }
  else { if (ps_contexto == "Producto") { return "cestaCompra,
     comentariosClientes"; }
    else { if (ps_contexto == "clientes") { return "
       comentariosClientes"; }
      else { return "-1"; }  }  }        }
```

- `recuperarEnlaceServicio`

```
public String recuperarEnlaceOperacion(String ps_servicio){
  if (ps_servicio == "anyadirCesta") { return "cestaCompra"; }
  else { return "-1"; } }
```

A.2.5 Soporte a la Presentación

Finalmente, se describe la aplicación del algoritmo propuesto (ver el algoritmo 11 en la página 135) sobre el caso de estudio. Este algoritmo se aplica para identificar, a partir del mapa navegacional, las operaciones a publicar en el servicio Web. Para cada uno de los pasos descritos se enumera la línea del algoritmo.

Paso 1. Comienza el algoritmo.

1: Existen operaciones en todos los grupos: *Lógica de la aplicación*, *Gestión de Usuarios*, *Recuperación de Información* y *Soporte a la Presentación*, por lo tanto devuelve verdadero

2: Se selecciona la operación `presentacionDeInformacion` para ser publicada en el servicio Web

Paso 2. Se comienza el recorrido por el primer contexto del mapa navegacional (ver el paso 2 en la figura A.14).

5: Se recorre el primer contexto del mapa navegacional, *Home*

6: Se selecciona la operación `presentacionDeHome` para ser publicada en el servicio Web

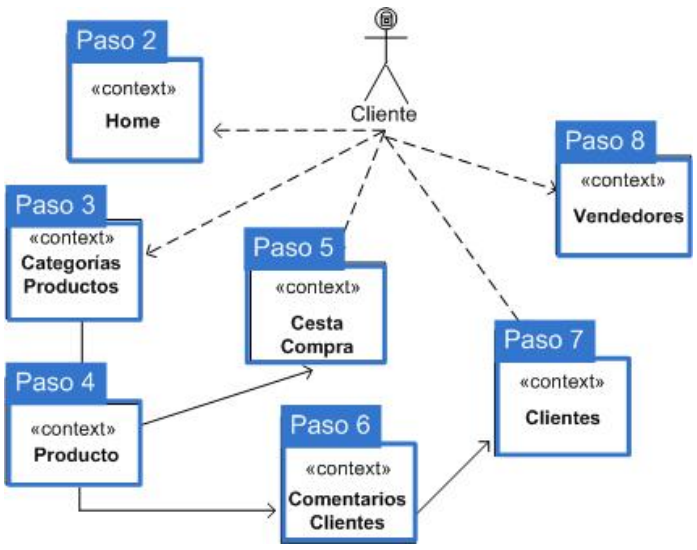

Figura A.14: Pasos del 2 al 9 del recorrido del mapa navegacional

Paso 3. Se continúa analizando el segundo contexto del mapa, el contexto *Categorías Productos* (ver el paso 3 en la figura A.14).

5: Se recorre el segundo contexto

6: Se selecciona la operación `presentacionDeCategorias-Productos` para ser publicada en el servicio Web

Paso 4. Se continúa analizando el tercer contexto del mapa, el contexto *Producto* (ver el paso 4 en la figura A.14).

5: Se recorre el tercer contexto

6: Se selecciona la operación `presentacionDeProductos` para ser publicada en el servicio Web

Paso 5. Se continúa analizando el cuarto contexto del mapa, el contexto *Cesta Compra* (ver el paso 5 en la figura A.14).

5: Se recorre el cuarto contexto

6: Se selecciona la operación `presentacionDeCestaCompra` para ser publicada en el servicio Web

Paso 6. Se continúa analizando el quinto contexto del mapa, el contexto *Comentarios Clientes* (ver el paso 6 en la figura A.14).

5: Se recorre el quinto contexto

6: Se selecciona la operación `presentacionDeComentarios-Clientes` para ser publicada en el servicio Web

Paso 7. Se continúa analizando el sexto contexto del mapa, el contexto *Clientes* (ver el paso 7 en la figura A.14).

5: Se recorre el sexto contexto

6: Se selecciona la operación `presentacionDeClientes` para ser publicada en el servicio Web

Paso 8. Se continúa analizando el último contexto del mapa, el contexto *Vendedores* (ver el paso 8 en la figura A.14).

5: Se recorre el séptimo y último contexto

6: Se selecciona la operación `presentacionDeVendedores` para ser publicada en el servicio Web

9: Finaliza la ejecución y devuelve el conjunto de operaciones identificadas

En la figura A.15 se muestran las operaciones y desde qué elementos, de los mapas navegacionales, han sido identificadas tras realizar la traza del algoritmo.

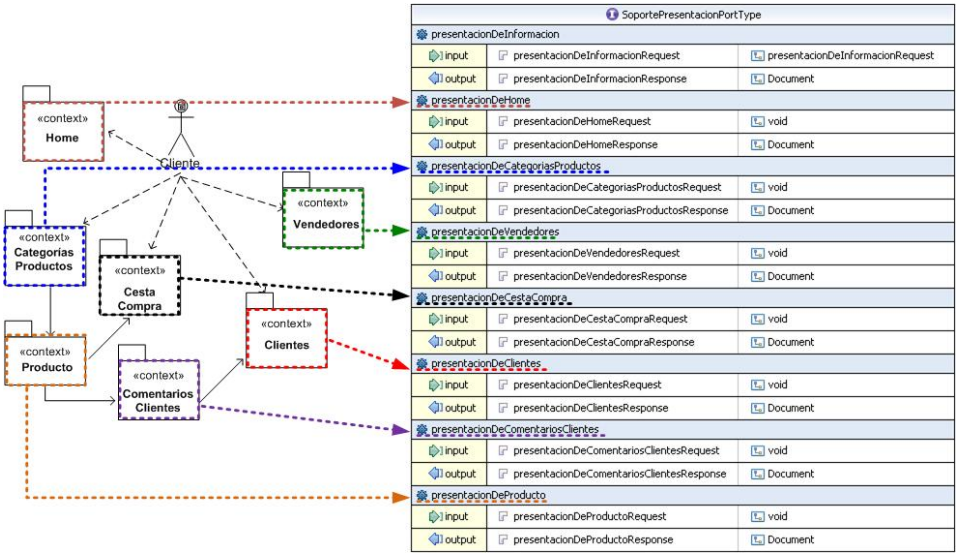

Figura A.15: Operaciones para el Soporte a la Presentación

Código de las operaciones

A continuación, para cada una de las operaciones identificadas, se detalla el código Java que las implementa. Cabe recordar que este grupo funcional no utiliza ninguna implementación generada por OlivaNOVA.

- `presentacionDeInformacion`

```
public HTMLDocument presentacionDeInformacion (String
    ps_operacion, String[] ps_parametrosOperacion, String
    ps_paginacion, String ps_orden, String ps_instancias){
  //Devuelve los datos de un Servicio Web con formato
  HTMLDocument lhd_presentacionDeInformacion = null;
  try {
    lhd_presentacionDeInformacion.open();
```

```
    lhd_presentacionDeInformacion.setTitle(``Generado con
       DISWOOM'');
    Document ld_datos = ps_operacion(ps_parametrosOperacion);
    HTMLTableElement lte_tabla = null;
    Node ln_dato = ld_datos.getFirstChild();
    while (ln_dato != null){
      lte_tabla.appendChild(ln_dato);
      ln_dato = ld_datos.getNextSibling(); }
    lhd_presentacionDeInformacion.appendChild(lte_tabla);
    lhd_presentacionDeInformacion.close();
  } catch(Exception e){ e.getMessage(); }
  return lhd_presentacionDeInformacion; }
```

- presentacionDeHome

```
public HTMLDocument presentacionDeHome(){
  //Devuelve una pagina completa de un contexto
  HTMLDocument lhd_Home = null;
  try {
    lhd_Home.open();
    lhd_Home.setTitle(``Generado con DISWOOM'');
    Document ld_datos = swRI.recuperarHome(null);
    HTMLTableElement lte_tabla = null;
    Node ln_dato = ld_datos.getFirstChild();
    while (ln_dato != null){
      lte_tabla.appendChild(ln_dato);
      ln_dato = ld_datos.getNextSibling(); }
    lhd_Home.appendChild(lte_tabla);
    String ls_Enlaces = swSN.recuperarEnlacesExploracion();
    lhd_Home.writeln(ls_Enlaces);
    lhd_Home.close();
  } catch(Exception e){ e.getMessage(); }
  return lhd_Home; }
```

- presentacionDeCategoriasProductos

```
public HTMLDocument presentacionDeCategoriasProductos(){
  //Devuelve una pagina completa de un contexto
  HTMLDocument lhd_CategoriasProductos = null;
  try {
    lhd_CategoriasProductos.open();
    lhd_CategoriasProductos.setTitle(``Generado con DISWOOM'');
    Document ld_datos = swRI.recuperarCategoriasProductos(null)
       ;
    HTMLTableElement lte_tabla = null;
    Node ln_dato = ld_datos.getFirstChild();
    while (ln_dato != null){
      lte_tabla.appendChild(ln_dato);
      ln_dato = ld_datos.getNextSibling(); }
    lhd_CategoriasProductos.appendChild(lte_tabla);
    String ls_Enlaces = swSN.recuperarEnlacesExploracion();
    lhd_CategoriasProductos.writeln(ls_Enlaces);
    lhd_CategoriasProductos.close();
  } catch(Exception e){ e.getMessage(); }
  return lhd_CategoriasProductos;       }
```

- presentacionDeProducto

```
public HTMLDocument presentacionDeProducto(String ps_Filtro){
  //Devuelve una página completa de un contexto
  HTMLDocument lhd_Producto = null;
  try {
    lhd_Producto.open();
    lhd_Producto.setTitle(``Generado con DISWOOM'');
    Document ld_datos = swRI.recuperarProducto(null);
    HTMLTableElement lte_tabla = null;
    Node ln_dato = ld_datos.getFirstChild();
    while (ln_dato != null){
      lte_tabla.appendChild(ln_dato);
      ln_dato = ld_datos.getNextSibling(); }
    lhd_Producto.appendChild(lte_tabla);
    String ls_Enlaces = swSN.recuperarEnlacesExploracion();
    lhd_Producto.writeln(ls_Enlaces);
    lhd_Producto.close();
  } catch(Exception e){ e.getMessage(); }
  return lhd_Producto; }
```

- presentacionDeCestaCompra

```
public HTMLDocument presentacionDeCestaCompra(){
  //Devuelve una pagina completa de un contexto
  HTMLDocument lhd_CestaCompra = null;
  try {
    lhd_CestaCompra.open();
    lhd_CestaCompra.setTitle(``Generado con DISWOOM'');
    Document ld_datos = swRI.recuperarCestaCompra(null);
    HTMLTableElement lte_tabla = null;
    Node ln_dato = ld_datos.getFirstChild();
    while (ln_dato != null){
      lte_tabla.appendChild(ln_dato);
      ln_dato = ld_datos.getNextSibling(); }
    lhd_CestaCompra.appendChild(lte_tabla);
    String ls_Enlaces = swSN.recuperarEnlacesExploracion();
    lhd_CestaCompra.writeln(ls_Enlaces);
    lhd_CestaCompra.close();
  } catch(Exception e){ e.getMessage(); }
  return lhd_CestaCompra;       }
```

- presentacionDeComentariosClientes

```
public HTMLDocument presentacionDeComentariosClientes(){
  //Devuelve una pagina completa de un contexto
  HTMLDocument lhd_ComentariosClientes = null;
  try {
    lhd_ComentariosClientes.open();
    lhd_ComentariosClientes.setTitle(``Generado con DISWOOM'');
    Document ld_datos = swRI.recuperarComentariosClientes(null)
       ;
    HTMLTableElement lte_tabla = null;
    Node ln_dato = ld_datos.getFirstChild();
    while (ln_dato != null){
      lte_tabla.appendChild(ln_dato);
      ln_dato = ld_datos.getNextSibling(); }
      lhd_ComentariosClientes.appendChild(lte_tabla);
      String ls_Enlaces = swSN.recuperarEnlacesExploracion();
      lhd_ComentariosClientes.writeln(ls_Enlaces);
      lhd_ComentariosClientes.close();
    } catch(Exception e){ e.getMessage(); }
    return lhd_ComentariosClientes;     }
```

- presentacionDeClientes

```
public HTMLDocument presentacionDeClientes(){
  //Devuelve una pagina completa de un contexto
  HTMLDocument lhd_Clientes = null;
  try {
    lhd_Clientes.open();
    lhd_Clientes.setTitle(``Generado con DISWOOM'');
    Document ld_datos = swRI.recuperarClientes(null);
    HTMLTableElement lte_tabla = null;
    Node ln_dato = ld_datos.getFirstChild();
    while (ln_dato != null){
      lte_tabla.appendChild(ln_dato);
      ln_dato = ld_datos.getNextSibling(); }
    lhd_Clientes.appendChild(lte_tabla);
    String ls_Enlaces = swSN.recuperarEnlacesExploracion();
    lhd_Clientes.writeln(ls_Enlaces);
          lhd_Clientes.close();
        } catch(Exception e){ e.getMessage(); }
        return lhd_Clientes; }
```

- presentacionDeVendedores

```
public HTMLDocument presentacionDeVendedores(){
  //Devuelve una pagina completa de un contexto
  HTMLDocument lhd_Vendedores = null;
  try {
    lhd_Vendedores.open();
    lhd_Vendedores.setTitle(´´Generado con DISWOOM'');
    Document ld_datos = swRI.recuperarVendedores(null);
    HTMLTableElement lte_tabla = null;
    Node ln_dato = ld_datos.getFirstChild();
    while (ln_dato != null){
      lte_tabla.appendChild(ln_dato);
      ln_dato = ld_datos.getNextSibling(); }
    lhd_Vendedores.appendChild(lte_tabla);
    String ls_Enlaces = swSN.recuperarEnlacesExploracion();
    lhd_Vendedores.writeln(ls_Enlaces);
          lhd_Vendedores.close();
        } catch(Exception e){ e.getMessage(); }
        return lhd_Vendedores; }
```

Apéndice B

Metamodelos

En este apéndice se presentan los dos metamodelos utilizados en el desarrollo de esta tesis. Estos metamodelos están definidos de acuerdo a MOF (Meta Object Facilities) [14]. Primero se presenta el metamodelo de la Especificación de Requisitos y después el de OOWS.

B.1 Metamodelo de la Especificación de Requisitos

En este metamodelo se encuentran representados los modelos que se han utilizado en el desarrollo de esta tesis (ver figura B.1): modelo de tareas y descripción de las mismas.

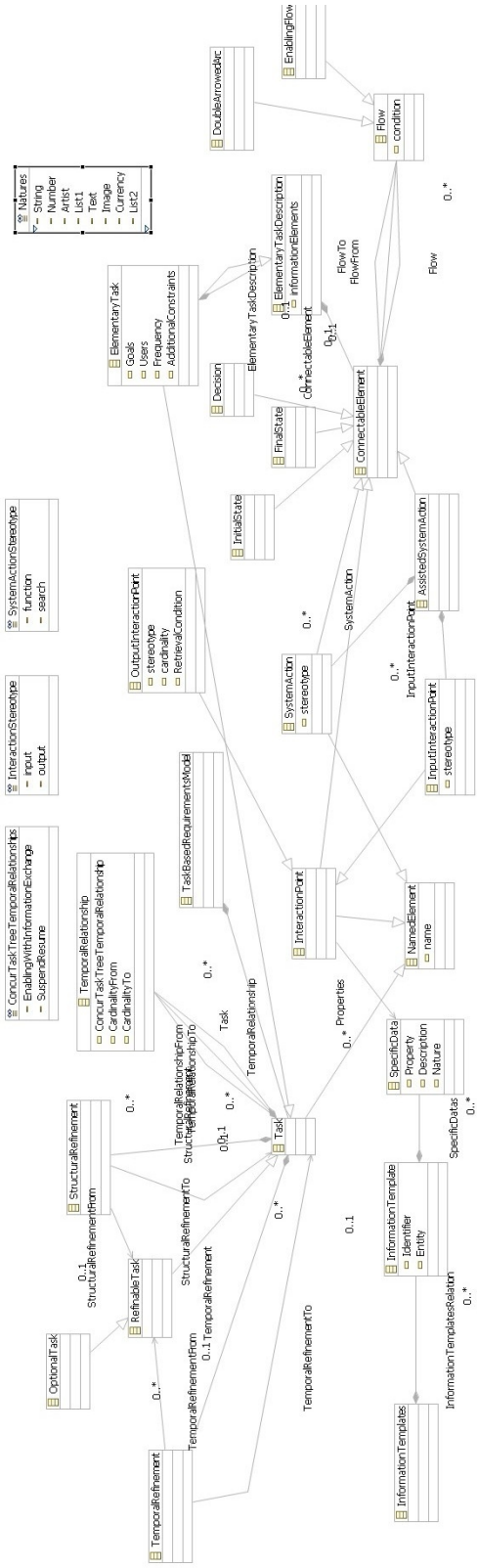

Figura B.1: Metamodelo ecore de la Especificación de Requisitos

B.2 Metamodelo de OOWS

En este metamodelo se encuentran representados los modelos que se han utilizado en el desarrollo de esta tesis (ver figura B.2): modelo de objetos, modelo de usuarios, modelo de navegación y modelo de presentación.

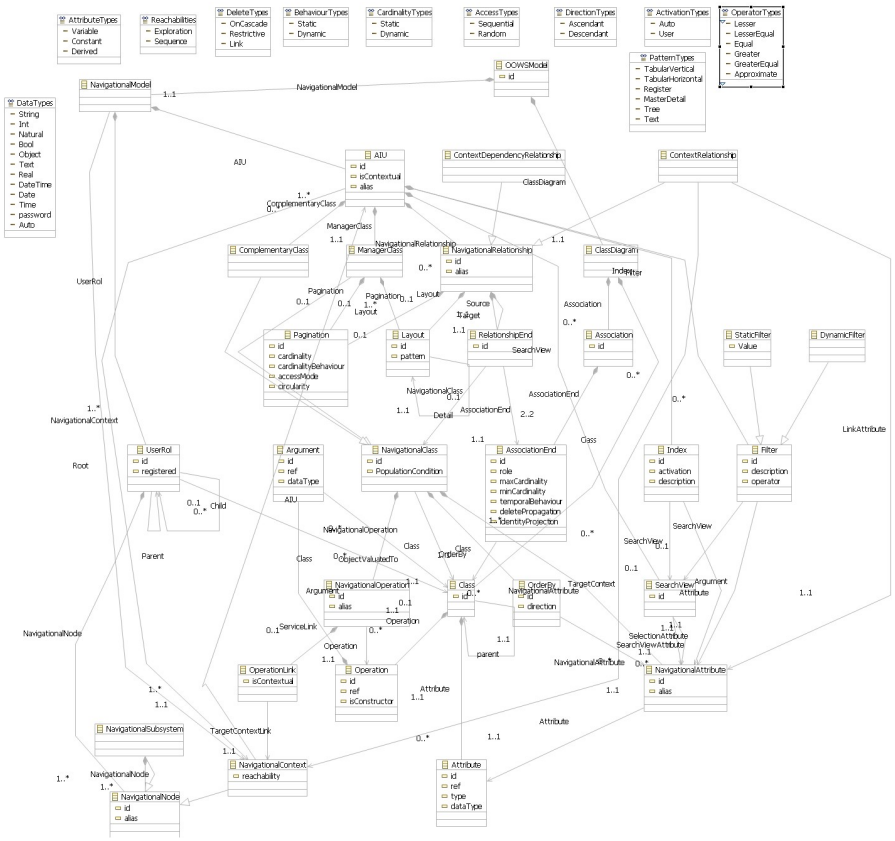

Figura B.2: Metamodelo ecore de OOWS

Apéndice C

Abreviaturas

ADVs Abstract Data Views

API Application Program Interface

CERN Consejo Europeo para la Investigación Nuclear

CIM Computational Independet Model

CRBAC Core RBAC

CSS Cascading Style Sheets

CTT Concur Task Trees

DSL Domain Specific Language

DSDM Software Dirigido por Modelos

HCI Human-Computer Interaction

HRBAC Hierarchical RBAC

HTML Hypertext Markup Language

IEEE Institute of Electrical and Electronics Engineers

ISO International Organization for Standardization

J2EE Java 2 Platform, Enterprise Edition

MDA Model Driven Architecture

MDD Model Driven Development

MDE Model Driven Engineering

MIDAS Metodología para el Desarrollo de Aplicaciones Web

MVC Model View Controller

.NET Plataforma de desarrollo de software de Microsoft

NIST National Institute of Standards and Technology

OOAD Object-Oriented Analysis and Design

OASIS Organization for the Advancement of Structured Information Standards

OMG Object Management Group

OO Object Oriented - Orientado a Objetos

OO-H Object Oriented Hypermedia Method

OOHDM Object-Oriented Hypermedia Design Method

OO-Method Object-Oriented Method

OOWS Object Oriented Web Solution

PI Punto de Interacción

PIM Platform Independent Model

PSM Platform Specific Model

RBAC Role Based Access Control

SHDM Semantic Hypermedia Design Method

SOA Service Oriented Architecture

SOAP Simple Access Protocol

SOC Service Oriented Computing

SPEM Software Process Engineering Metamodel

UAI Unidad Abstracta de Información

UDDI Universal, Description, Discovery and Integration

UML Unified Modeling Language

UWE UML Based Web Engineering

W3C World Wide Web Consortium

WebML Web Modeling Language

WSA Web Service Architecture

WSDL Web Service Description Language

XML Extender Mark up Language

Referencias

[1] Dirk Krafzig, Karl Banke, and Dirk Slama. *Enterprise SOA: Service Oriented Architecture Best Practices*. Prentice Hall Professional Technical Reference, Upper Saddle Rive, 2005. [*Citado en la página 1.*]

[2] Mike P. Papazoglou. Service-oriented computing: concepts, characteristics and directions. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering, 2003. WISE 2003*, pages 3–12, 2003. [*Citado en las páginas 1 y 20.*]

[3] Mike P. Papazoglou and Willem-Jan van den Heuvel. Web services management: a survey. *IEEE Internet Computing*, 9(6):58–64, 2005. [*Citado en la página 1.*]

[4] D. Sprott, L. Wilkes, R. Veryard, and J. Stephenson. Web services roadmap. Technical report, CBI Report Series, 2003. [*Citado en la página 1.*]

[5] W3C. Extensible markup language (xml). [*Citado en las páginas 1 y 25.*]

[6] W3C. Soap version 1.2 part 1: Messaging framework (second edition), 2007. [*Citado en las páginas 1 y 26.*]

[7] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web service description language, March 2001. [*Citado en las páginas 1, 27 y 44.*]

[8] UDDI Version 3.0.2. Uddi spec technical committee draft, dated 20041019, 2004. [*Citado en las páginas 1 y 28.*]

[9] San Murugesan Chair. Web engineering. *SIGWEB Newsl.*, 8(3):28–32, 1999. [*Citado en las páginas 2 y 36.*]

[10] S.J. Mellor, A.N. Clark, and T. Futagamii. Model-driven development - guest editor's introduction. *IEEE Software*, 20(5):14–18, Sept.-Oct. 2003. [*Citado en las páginas 2, 4, 5 y 54.*]

[11] Oscar Pastor, Jaime Gómez, Emilio Insfrán, and Vicente Pelechano. The oo-method approach for information systems modelling: From object-oriented conceptual modeling to automated programming. *Information Systems*, 26:507–534, 2001. [*Citado en la página 2.*]

[12] Joan Fons. *OOWS: Un Mètode Dirigit per Models per al Desenvolupament d"Aplicacions Web.* PhD thesis, Universitat Politècnica de València, 2008. [*Citado en las páginas 2, 54, 64, 66, 73, 74, 141, 144 y 191.*]

[13] Joan Fons, Vicente Pelechano, Oscar Pastor, Pedro Valderas, and Victoria Torres. *Web Engineering: Modelling and Implementing Web Applications*, chapter Applying the Oows Model-Driven Approach for Developing Web Applications. The Internet Movie Database Case Study, pages 65–108. Volume 1 of Rossi et al. [122], 2008. [*Citado en la página 2.*]

[14] Object Management Group. Meta object facilities (mof) 2.0. http://www.omg.org. [*Citado en las páginas 3 y 243.*]

[15] Jon Oldevik. Mofscript user guide. version 0.5 (mofscript v 1.1.7). http://www.omg.org, June 2006. [*Citado en las páginas 3 y 190.*]

[16] Mike P. Papazoglou and Jian Yang. Design methodology for web services and business processes. In *TES '02: Proceedings of the Third International Workshop on Technologies for E-Services*, pages 54–64, London, UK, 2002. Springer-Verlag. [*Citado en la página 3.*]

[17] Karim Baïna, Boualem Benatallah, Fabio Casati, and Farouk Toumani. Advanced information systems engineering. *Lecture Notes in Computer Science*, 3084/2004:290–306, 2004. [*Citado en la página 3.*]

[18] Bart Orriëns, Jian Yang, and Mike P. Papazoglou. Model driven service composition. In Maria E. Orlowska, Sanjiva Weerawarana, and Et, editors, *ICSOC*, volume 2910 of *LNCS*, 2003. [*Citado en la página 3.*]

[19] Marco Brambilla. Extending hypertext conceptual models with process-oriented primitives. In Il-Yeol Song, Stephen W. Liddle, Tok Wang Ling, and Peter Scheuermann, editors, *ER*, volume 2813 of *Lecture Notes in*

Computer Science, pages 246–262. Springer, 2003.
[*Citado en las páginas 3 y 43.*]

[20] Marco Brambilla, Stefano Ceri, Sara Comai, Piero Fraternali, and Iona Manolescu. Specification and design of workflow-driven hypertexts. In *In WWW (Posters)*, 2003. [*Citado en la página 3.*]

[21] Hans Albrecht Schmid and Gustavo Rossi. Modeling and designing processes in e-commerce applications. *IEEE Internet Computing*, 8(1):19–27, 2004. [*Citado en la página 3.*]

[22] Nora Koch, Andreas Kraus, Cristina Cachero, and Santiago Meliá. Integration of business processes in web application models. *Journal of Web Engineering*, 3:22–49, 2004. [*Citado en las páginas 3 y 4.*]

[23] Alexander Knapp, Nora Koch, Gefei Zhang, and Hanns-Martin Hassler. Modeling business processes in web applications with argouwe. *Lecture Notes in Computer Science*, 3276:69–83, 2004. [*Citado en la página 3.*]

[24] James Snell, Doug Tidwell, and Pavel Kulchenko. *Programming Web services with SOAP*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002. [*Citado en la página 4.*]

[25] Alex Ferrara and Matthew MacDonald. *Programming .NET Web Services*. O'Reilly, 2002. [*Citado en la página 4.*]

[26] Inderjeet Singh, Sean Brydon, Greg Murray, Vijay Ramachandran, Thierry Violleau, and Beth Stearns. *Designing Web services with the J2EE(TM) 1.4 Platform: JAX-RPC, SOAP, and XML Technologies*. Addison Wesley Professional, 2004. [*Citado en la página 4.*]

[27] IBM Redbooks publication. Web services handbook for websphere application server 6.1, October 2006. [*Citado en la página 4.*]

[28] Flavius Frasincar, Geert-Jan Houben, and Peter Barna. Building web information systems using web services. In *In Databases and Information Systems*, pages 187 – 193, 7th International Baltic Conference, 2006.
[*Citado en las páginas 4, 37 y 38.*]

[29] Hans Albrecht Schmid. A service-centric architecture for web applications. In *ICWE*, pages 575–580, 2005. [*Citado en las páginas 4, 41 y 85.*]

[30] Ioana Manolescu, Marco Brambilla, Stefano Ceri, Sara Comai, and Piero Fraternali. Model-driven design and deployment of service-enabled web applications. *ACM Trans. Inter. Tech.*, 5(3):439–479, 2005. [*Citado en las páginas 4, 44 y 45.*]

[31] Kenji Takahashi and Eugene Liang. Analysis and design of web-based information systems. *Comput. Netw. ISDN Syst.*, 29(8-13):1167–1180, 1997. [*Citado en la página 4.*]

[32] Colin Atkinson and Thomas Kühne. Model-driven development: A metamodeling foundation. *IEEE Softw.*, 20(5):36–41, 2003. [*Citado en la página 5.*]

[33] Bran Selic. The pragmatics of model-driven development. *IEEE Softw.*, 20(5):19–25, 2003. [*Citado en la página 5.*]

[34] Object Management Group. Software process engineering meta-model, version 2.0, 2008. [*Citado en las páginas 7 y 52.*]

[35] Salvatore T. March and Gerald F. Smith. Design and natural science research on information technology. *Decision Support Systems*, 15(4):251–266, 1995. [*Citado en la página 9.*]

[36] A.R. Hevner, S.T. March, J. Park, and S. Ram. Design science in information systems research. *Management information systems quarterly*, 28(1):75–106, 2004. [*Citado en la página 9.*]

[37] BEA Systems. Enterprise portal rationalization: using a service-oriented architecture to stop web asset sprawl. Technical report, BEA Systems, 2004. [*Citado en la página 16.*]

[38] C. H. Crawford, G. P. Bate, L. Cherbakov, K. Holley, and C. Tsocanos. Toward an on demand service-oriented architecture. *IBM SYSTEMS JOURNAL*, 44(1):81–107, 2005. [*Citado en la página 16.*]

[39] G. Phifer. The fifth generation of portals supports soa and process integration. *Gartner Report*, 2006. [*Citado en la página 16.*]

[40] W3C Working Group. Web services architecture, February 2004. http://www.w3.org/TR/ws-arch/. [*Citado en las páginas 16 y 86.*]

[41] W3C. Guías breves de tecnologías w3c. [*Citado en la página 19.*]

[42] Romin Irani. Practical considerations in implementing web services, 2001. [*Citado en la página 20.*]

[43] Victoria Torres, Ricardo Quintero, Marta Ruiz, and Vicente Pelechano. Towards the integration of data and functionality in web applications. a model driven approach. In Orlando Belo, Johann Eder, João Falcão e Cunha, and Oscar Pastor, editors, *CAiSE Short Paper Proceedings*, volume 161 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005. [*Citado en la página 22.*]

[44] Victoria Torres. *A Web Engineering Approach for the Development of Business-Process Driven Web Applications*. PhD thesis, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, 2008. [*Citado en la página 22.*]

[45] Ricardo Quintero. *Desarrollo Dirigido por Modelos de Aplicaciones Web que integran Datos y Funcionalidad a partir de Servicios Web*. PhD thesis, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, 2007. [*Citado en las páginas 22 y 201.*]

[46] Ethan Cerami. *Web Services Essentials*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002. [*Citado en la página 24.*]

[47] ANSI. Role-based access control. *American National Standard for Information technology. Incits 359*, 2004. [*Citado en las páginas 29, 102 y 105.*]

[48] David Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed nist standard for role-based access control. *ACM Trans. Inf. Syst. Secur.*, 4(3):224–274, 2001. [*Citado en la página 29.*]

[49] Roger S. Pressman, Ted Lewis, Ben Adida, Ellen Ullman, Tom DeMarco, Thomas Gilb, Brent Gorda, Watts Humphrey, and Ray Johnson. Can internet-based applications be engineered? *IEEE Softw.*, 15(5):104–110, 1998. [*Citado en la página 36.*]

[50] Athula Ginige and San Murugesan. Guest editors' introduction: Web engineering&#151 an introduction. *IEEE MultiMedia*, 8(1):14–18, 2001. [*Citado en la página 36.*]

[51] G. Kappel, B. Pröll, S. Reich, W. Retschitzegger, P. Grünbacher, W. Schwinger, N. Koch, C. Eichinger, G. Austaller, A. Hartl, M. Lauff, F. Lyardet, M. Mühlhäuser, M. Nussbaumer, M. Gaedke, C. Steindl, R. Ramler, J. Altmann, A. Ebner, B. Pröll, H. Werthner, H. Mayr, G. Engels, M. Lohmann, A. Wagner, M. Hitz, G. Leitner, R. Melcher, G. Kotsis, M. Wimmer, A. Kemper, S. Seltzsam, W. Behrendt, and N. Arora. Web engineering. the discipline of systematic development of web applications. *John Wiley & Sons,*, 2006. [*Citado en la página 36.*]

[52] Emilia Mendes and Nile Mosley. *Web Engineering*. Springer Berlin Heidelberg, 2006. [*Citado en la página 36.*]

[53] Peter Barna and Geert-Jan Houben. User interaction in modern web information systems. In *In P.M.E. De Bra (Ed.), Proceedings Conferentie Informatiewetenschap*, pages 19–28, 2003.
[*Citado en las páginas 36 y 37.*]

[54] Jaime Gómez, Cristina Cachero, and Oscar Pastor. Extending a conceptual modelling approach to web application design. In *Conference on Advanced Information Systems Engineering*, pages 79–93, 2000.
[*Citado en las páginas 36 y 39.*]

[55] Daniel Schwabe and Gustavo Rossi. An object oriented approach to web-based applications design. *Theory and Practice of Object Systems*, 4(4):207–225, 1998. [*Citado en las páginas 36 y 40.*]

[56] Nora Koch and Martin Wirsing. Software engineering for adaptive hypermedia applications, 2001. [*Citado en las páginas 36 y 42.*]

[57] Stefano Ceri, Piero Fraternali, and Aldo Bongio. Web modeling language (webml): a modeling language for designing web sites. *Comput. Networks*, 33(1-6):137–157, 2000. [*Citado en las páginas 36 y 43.*]

[58] O. M. F. De Troyer and C. J. Leune. Wsdm: a user centered design method for Web sites. *Computer Networks and ISDN Systems*, 30(1–7):85–94, 1998. [*Citado en las páginas 36 y 45.*]

[59] Geert-Jan Houben, Kees van der Sluijs, Peter Barna, Jeen Broekstra, Sven Casteleyn, Zoltàn Fiala, and Flavius Frasincar. *Web Engineering: Modelling and Implementing Web Applications*, chapter HERA, pages

263–301. Volume 1 of Rossi et al. [122], 2008.
[*Citado en las páginas 37 y 85.*]

[60] Jaime Gómez and Cristina Cachero. Oo-h method: extending uml to model web interfaces. *Idea Group Publishing*, pages 144–173, 2003.
[*Citado en la página 39.*]

[61] Oscar Pastor, Emilio Insfrán, Vicente Pelechano, José Romero, and José Merseguer. Oo-method: An oo software production environment combining conventional and formal methods. In *CAiSE '97: Proceedings of the 9th International Conference on Advanced Information Systems Engineering*, pages 145–158, London, UK, 1997. Springer-Verlag.
[*Citado en las páginas 39, 54, 64 y 65.*]

[62] Irene Garrigós, Jaime Gómez, Peter Barna, and Geert-Jan Houben. A reusable personalization model in web application design. In *In 2nd Workshop on Web Information Systems Modelling (WISM 2005)*, 2005.
[*Citado en la página 39.*]

[63] Gustavo Rossi and Daniel Schwabe. *Web Engineering: Modelling and Implementing Web Applications*, chapter Modeling and Implementing Web Applications with Oohdm, pages 109–155. Volume 1 of Rossi et al. [122], 2008. [*Citado en las páginas 40 y 42.*]

[64] Franca Garzotto, Paolo Paolini, and Daniel Schwabe. Hdm—a model-based approach to hypertext application design. *ACM Trans. Inf. Syst.*, 11(1):1–26, 1993. [*Citado en la página 40.*]

[65] Fernanda Lima and Daniel Schwabe. Modeling applications for the semantic web. In Juan Manuel Cueva Lovelle, Bernardo Martín González Rodríguez, Luis Joyanes Aguilar, José Emilio Labra Gayo, and María del Puerto Paule Ruíz, editors, *ICWE*, volume 2722 of *Lecture Notes in Computer Science*, pages 417–426. Springer, 2003.
[*Citado en la página 41.*]

[66] Patricia Vilain, Daniel Schwabe, and Clarisse Sieckenius de Souza. A diagrammatic tool for representing user interaction in UML. In Andy Evans, Stuart Kent, and Bran Selic, editors, *UML 2000 - The Unified Modeling Language. Advancing the Standard. Third International Conference, York, UK, October 2000, Proceedings*, volume 1939, pages 133–147. Springer, 2000. [*Citado en la página 41.*]

[67] Donald D. Cowan and Carlos J. P. Lucena. Abstract data views: An interface specification concept to enhance design for reuse. *IEEE Transactions on Software Engineering*, 21(3):229–243, 1995.
[*Citado en la página 41.*]

[68] Nora Koch, Gefei Zhang, and María José Escalona. Model transformations from requirements to web system design. In *ICWE '06: Proceedings of the 6th international conference on Web engineering*, pages 281–288, New York, NY, USA, 2006. ACM. [*Citado en la página 42.*]

[69] Nora Koch, Alexander Knapp, Gefei Zhang, and Hubert Baumeister. *Web Engineering: Modelling and Implementing Web Applications*, chapter Uml-Based Web Engineering. An Approach Based on Standards, pages 157–191. Volume 1 of Rossi et al. [122], 2008.
[*Citado en las páginas 42 y 43.*]

[70] Andreas Kraus, Alexander Knapp, and Nora Koch. Model-driven generation of web applications in uwe. In Nora Koch, Antonio Vallecillo, and Geert-Jan Houben, editors, *Proc. 3rd Int. Wsh. Model-Driven Web Engineering (MDWE'07)*, volume 261 of *CEUR-WS*, 2007.
[*Citado en la página 43.*]

[71] Stefano Ceri, Piero Fraternali, Aldo Bongio, Marco Brambilla, Sara Comai, and Maristella Matera. *Designing Data-Intensive Web Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.
[*Citado en la página 43.*]

[72] Marco Brambilla, Sara Comai, Piero Fraternali, and Maristella Matera. *Web Engineering: Modelling and Implementing Web Applications*, chapter Designing Web Applications with Webml and Webratio, pages 221–261. Volume 1 of Rossi et al. [122], 2008.
[*Citado en las páginas 43, 44, 45 y 85.*]

[73] Vicente Pelechano, Marta Ruiz, Joan Fons, and Pedro Valderas. *Avances en Comercio Electrónico*, chapter Desarrollo de Aplicaciones WEB basadas en Servicios WEB XML, pages 99–117. Francisco J. García Peñalvo, 2002. [*Citado en la página 44.*]

[74] Marta Ruiz, Ausiàs Armesto, and Vicente Pelechano. Hacia la generación de aplicaciones web en arquitecturas soa. una aproximación basada en

mda. In Ernesto Pimentel Juan Hernandez, editor, *Jornadas de Ingeniería de Software y Base de Datos (JISBD*, pages 531 – 538, 2004. [*Citado en la página 44.*]

[75] Marco Brambilla, Stefano Ceri, Sara Comai, and Piero Fraternali. A case tool for modelling and automatically generating web service-enabled applications. *International Journal of Web Engineering and Technology*, 2(4):354 – 372, 2006. [*Citado en la página 45.*]

[76] Olga De Troyer, Sven Casteleyn, and Peter Plessers. *Web Engineering: Modelling and Implementing Web Applications*, chapter WSDM: Web Semantics Design Method, pages 303–351. Volume 1 of Rossi et al. [122], 2008. [*Citado en las páginas 45 y 46.*]

[77] Jehad Najjar, Abdalghani Mushtaha, and Olga De Troyer. Integrating business rules into the web development process. In *Proceedings of the 2nd International Conference on Information Technology - Human Computer Interaction (ICIT2005)*, pages 404–410, Amman, Jordan (2005), 2005. [*Citado en la página 46.*]

[78] Donald Firesmith and Brian Henderson-Sellers. *The OPEN Process Framework*. Addion-Wesley Longman, 2001. [*Citado en la página 52.*]

[79] Henrik Terävä. Software process modeling with eclipse process framework. Master's thesis, Department of Information Technology, Software Engineering, 2007. [*Citado en la página 52.*]

[80] Pierre-Yves Cunin, R. Mark Greenwood, Laurent Francou, Ian Robertson, and Brian Warboys. The pie methodology - concept and application. In *EWSPT '01: Proceedings of the 8th European Workshop on Software Process Technology*, pages 3–26, London, UK, 2001. Springer-Verlag. [*Citado en la página 52.*]

[81] Object Management Group. Unified modelling language 2.0. [*Citado en las páginas 53 y 62.*]

[82] Pedro Valderas. *A Requirements Engineering Approach for the Development of Web Applications*. PhD thesis, Department of Information Systems and Computation Technical University of Valencia, Marzo 2008. [*Citado en las páginas 58, 59 y 191.*]

[83] Olga De Troyer and Sven Castel. Modeling complex processes for web applications using wsdm. In *Third International Workshop on Web-Oriented Software Technology*, 2003. [*Citado en la página 58.*]

[84] D. Sinnig, P. Forbrig, and A. Seffah. Patterns in model-based development. In *CEUR Workshop Proceedings*, 2004. [*Citado en la página 58.*]

[85] Tassanee Er-Jongmanee. Xml-driven device independent user interface build rich client applications using xml. Master's thesis, Rheinisch-Westfälische Technische Hochschule Aachen, 2005. [*Citado en la página 58.*]

[86] David Paquette and Kevin A. Schneider. Interaction templates for constructing user interfaces from task models. In *CADUI*, pages 221–232, 2004. [*Citado en la página 58.*]

[87] Pedro Valderas and Vicente Pelechano. Improving communication in requirements engineering activities for web applications. In Luciano Baresi, Piero Fraternali, and Geert-Jan Houben, editors, *ICWE*, volume 4607 of *Lecture Notes in Computer Science*, pages 242–247. Springer, 2007. [*Citado en la página 61.*]

[88] Fabio Paterno, Cristiano Mancini, and Silvia Meniconi. Concurtasktrees: A diagrammatic notation for specifying task models. In *INTERACT '97: Proceedings of the IFIP TC13 Interantional Conference on Human-Computer Interaction*, pages 362–369, London, UK, 1997. Chapman and Hall, Ltd. [*Citado en la página 61.*]

[89] Pedro Valderas, Joan Fons, and Vicente Pelechano. Using task descriptions for the specification of web application requirements. In João Araújo, Amador Durán Toro, and João Falcão e Cunha, editors, *WER*, pages 257–268, 2005. [*Citado en la página 61.*]

[90] Pedro Valderas, Joan Fons, and Vicente Pelechano. From web requirements to navigational design - a transformational approach. In David Lowe and Martin Gaedke, editors, *ICWE*, volume 3579 of *Lecture Notes in Computer Science*, pages 506–511. Springer, 2005. [*Citado en la página 61.*]

[91] CARE Technologies S.A. Olivanova modeller. [*Citado en las páginas 73, 74, 76, 141 y 191.*]

[92] CARE Technologies S.A. Olivanova transformation engine. [*Citado en las páginas 73, 74, 76, 141 y 192.*]

[93] Francisco Valverde. Diseño e implementación de un entorno mda para la producción de aplicaciones web. Master's thesis, Tesina del Master de Ingeniería del Software, Métodos Formales y Sistemas de Información, 2007. [*Citado en las páginas 73, 74, 141 y 144.*]

[94] Francisco Valverde and Pedro Valderas. Oows suite: Un entorno de desarrollo para aplicaciones web basado en mda. In *Workshop Iberoamericano de Ingenieria de Requisitos y Ambientes Software (IDEAS)*, 2007. [*Citado en la página 74.*]

[95] Tim O'Reilly. What is web 2.0. design patterns and business models for the next generation of software. *Social Science Research Network Working Paper Series*, 2005. [*Citado en la página 81.*]

[96] Christoph Schroth and Till Janner. Web 2.0 and soa: Converging concepts enabling the internet of services. *IT Professional*, 9(3):36–41, 2007. [*Citado en la página 81.*]

[97] Cesare Pautasso and Erik Wilde. Why is the web loosely coupled? a multi-faceted metric for service design. In *18th International World Wide Web Conference*, pages 911–920, April 2009. [*Citado en las páginas 86 y 202.*]

[98] Liam O'Brien, Paulo Merson, and Len Bass. Quality attributes for service-oriented architectures. In *SDSOA '07: Proceedings of the International Workshop on Systems Development in SOA Environments*, page 3, Washington, DC, USA, 2007. IEEE Computer Society. [*Citado en la página 86.*]

[99] Jen-Yao Chung, Kwei-Jay Lin, and Richard G. Mathieu. Web services computing: advancing software interoperability. *Computer*, 36(10):35–37, Oct 2003. [*Citado en la página 86.*]

[100] Lawrence Wilkes and Richard Veryard. Service-oriented architecture: Considerations for agile systems. *CBDI Forum*, April 2004. [*Citado en la página 86.*]

[101] Thomas Erl. *Soa: Principles of Service Design*. Prentice Hall PTR, Upper Saddle River, 2007. [*Citado en la página 88.*]

[102] BEA Web Logic Portal. Portlet developer's guide, 2007. [*Citado en las páginas 89 y 131.*]

[103] Anant Jhingran. Enterprise information mashups: integrating information, simply. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pages 3–4. VLDB Endowment, 2006. [*Citado en las páginas 89 y 131.*]

[104] Mark Allen Weiss. *Data structures and algorithm analysis*. Redwood City, California, 1995. [*Citado en la página 91.*]

[105] Marta Ruiz, Pedro Valderas, Victoria Torres, and Vicente Pelechano. A model driven approach to design web services in a web engineering method. In Orlando Belo, Johann Eder, João Falcão e Cunha, and Oscar Pastor, editors, *CAiSE Short Paper Proceedings*, volume 161 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005. [*Citado en la página 105.*]

[106] Marta Ruiz, Pedro Valderas, and Vicente Pelechano. Applying a web engineering method to design web services. In Boualem Benatallah, Fabio Casati, and Paolo Traverso, editors, *ICSOC*, volume 3826 of *Lecture Notes in Computer Science*, pages 576–581. Springer, 2005. [*Citado en la página 105.*]

[107] Marta Ruiz and Vicente Pelechano. Model driven design of web service operations using web engineering practices. In Cesare Pautasso and Christoph Bussler, editors, *Workshop on Emerging Web Services Technology (WEWST)*, volume 234, pages 75 – 90, Zurich (Suiza), December 2006. CEUR Workshop Proceedings. [*Citado en la página 105.*]

[108] Gonzalo Rojas, Vicente Pelechano, and Joan Fons. A model-driven approach to include adaptive navigational techniques in web applications. In *International Workshop on Web Oriented Software Technology*, pages 13 – 24, Porto (Portugal), June 2005. [*Citado en la página 124.*]

[109] Gonzalo Rojas and Vicente Pelechano. A methodological approach for incorporating adaptive navigation techniques into web applications. In Anne H. H. Ngu, Masaru Kitsuregawa, Erich J. Neuhold, Jen-Yao Chung,

and Quan Z. Sheng, editors, *WISE*, volume 3806 of *Lecture Notes in Computer Science*, pages 203–216. Springer, 2005.
[*Citado en la página 124.*]

[110] Gonzalo Rojas. *Modelling Adaptative Web Applications in OOWS*. Tesis doctoral en informática, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, 2008.
[*Citado en la página 124.*]

[111] Mehmet Altinel, Paul Brown, Susan Cline, Rajesh Kartha, Eric Louie, Volker Markl, Louis Mau, Yip-Hing Ng, David E. Simmen, and Ashutosh Singh. Damia - a data mashup fabric for intranet applications. In *VLDB*, pages 1370–1373, 2007. [*Citado en la página 131.*]

[112] Anupriya Ankolekar, Markus Krötzsch, Thanh Tran, and Denny Vrandecic. The two cultures: mashing up web 2.0 and the semantic web. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 825–834, New York, NY, USA, 2007. ACM Press.
[*Citado en la página 131.*]

[113] Declan Butler. Mashups mix data into global service. *Nature*, 439(7072):6–7, January 2006. [*Citado en la página 131.*]

[114] Patrick Holz. Mashups - motivation, organisation und geschäftsmodelle. *HMD - Praxis der Wirtschaftsinformatik*, 255:70–77, June 2007.
[*Citado en la página 131.*]

[115] Marwan Sabbouh, Jeff Higginson, Salim Semy, and Danny Gagne. Web mashup scripting language. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 1305–1306, New York, NY, USA, 2007. ACM. [*Citado en la página 131.*]

[116] Erich Gamma, Richar Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley, 1995. [*Citado en la página 144.*]

[117] BOE. Ley orgánica 15/1999, de 13 de diciembre, de protección de datos de carácter personal (lopd), Diciembre 1999. Rango: Ley Orgánica.
[*Citado en la página 160.*]

[118] Ministerio de Justicia (BOE n. 17 de 19/1/2008). Real decreto 1720/2007, de 21 de diciembre, por el que se aprueba el reglamento de desarrollo de la ley orgánica 15/1999, de 13 de diciembre, de protección de datos de carácter personal., Enero 2008. Rango: Real Decreto Páginas: 4103 - 4136 Referencia: 2008/00979. [*Citado en la página 160.*]

[119] Pedro Valderas, Vicente Pelechano, and Oscar Pastor. Introducing graphic designers in a web development process. In John Krogstie, Andreas L. Opdahl, and Guttorm Sindre, editors, *CAiSE*, volume 4495 of *Lecture Notes in Computer Science*, pages 395–408. Springer, 2007. [*Citado en la página 181.*]

[120] Francisco Valverde and Oscar Pastor. Dealing with rest services in model-driven web engineering methods. In *V Jornadas Científico - Técnicas en servicios Web y SOA (JSWEB)*, 2009. [*Citado en la página 201.*]

[121] Marta Ruiz. Gestión de usuarios y control de acceso basado en roles. un caso real. Master's thesis, Universidad Politécnica de Valencia, 2007. [*Citado en la página 201.*]

[122] Gustavo Rossi, Oscar Pastor, Daniel Schwabe, and Luis Olsina, editors. *Web Engineering: Modelling and Implementing Web Applications (Human-Computer Interaction Series)*, volume 1. Springer London, 2008. [*Citado en las páginas 252, 257, 258 y 259.*]

Sobre la autora

Marta Ruiz Server nació el 11 de abril de 1976 en Valencia, España.

En septiembre de 2002 terminó los estudios de Ingeniera Informática por la Universidad Politécnica de Valencia. Cursó sus estudios de doctorado con una beca FPI y estuvo vinculada al grupo de investigación OO-Method. Durante su estancia en la universidad (del año 2002 al 2006) desarrolló su actividad investigadora en el grupo OO-Method bajo la supervisión del doctor Vicente Pelechano. Sus publicaciones están firmadas con el nombre de Marta Ruiz. Además de la investigación, durante esos años dirigió varios proyectos final de carrera e impartió docencia.

En enero de 2007 comenzó a trabajar en Dimensión Informática, empresa que entró a formar parte de Indra en enero de 2008. Durante 2007 y 2008 trabajó como analista en el proyecto ITACA, Innovación Tecnológica Aplicada a Centros y Alumnos, para la Conselleria de Educación. En junio de 2008 obtuvo el título del máster en Ingeniería del Software, Métodos Formales y Sistemas de la Información por la UPV realizando un trabajo de fin de máster en el entorno de la empresa. Actualmente trabaja como Ingeniera del Software Senior en el proyecto SIA, Sistema Integral Ambulatorio, para la Conselleria de Sanidad.

Correo electrónico: marruise@gmail.com

UNIVERSIDAD
POLITECNICA
DE VALENCIA