Universidad Politécnica de Valencia

Departamento de Sistemas Informáticos y Computación

Tesis Doctoral

"Desarrollo Dirigido por Modelos de Aplicaciones Web que integran Datos y Funcionalidad a partir de Servicios Web"

Presentada por:
Ricardo Rafael Quintero Meza

Dirigida por:
Dr. Vicente Pelechano Ferragud

Valencia, España, marzo de 2008

Con todo mi amor para
Hidaelia (mi mocito),
Ricardo Isaac (mi chiquito papi)
y Diego Isaí (mi pollito),
porque su apoyo, sonrisas y besos
han dado fruto ...

Agradecimientos

"Mejor es el fin del negocio que el principio..."
Salomón (Eclesiastés 7:8, RV1960)

El producto final de esta intensa jornada académica es el resultado de la relación con muchas personas que estimo y aprecio de forma especial, quienes dedicaron un espacio de su vida para mi formación doctoral. Con riesgo de omitir alguna, hoy quiero agradecer:

Al Dr. Vicente Pelechano –Pele- quien con excelencia académica, profesionalismo impecable pero sobre todo con paciencia y gran sentido humano, fue formando y colocando en mí los fundamentos como doctor e investigador. Muchas lecciones aprendí con él que me resultaría sumo difícil enumerarlas, de todas ellas las que quizá han dejado huella más profunda son la búsqueda continua de claridad, sencillez y honestidad. No las olvidaré.

A mis amigos del DSIC, quienes aún en la distancia dieron apoyo a mi trabajo de investigación: Vicky Torres, Marta Ruiz, Javier Muñoz, Joan Fons, Pedro Valderas, Gonzalo Rojas, Nelly Condori, Isabel Díaz, Guadalupe Ramos, Rogelio Limón, Ricardo Blanco y Gustavo Arroyo. Un agradecimiento muy especial al Dr. Oscar Pastor, de quien no sólo aprendí los fundamentos de UML, sino también recibí su apoyo a lo largo de toda mi formación. Mi agradecimiento también a todos mis profesores del DSIC.

Al Dr. Javier Ortiz del CENIDET, quien fue el primer contacto que tuve para considerar el ingreso a este programa doctoral.

A Leopoldo y Clemente, con quienes compartí momentos de amistad y compañerismo necesarios para sobrellevar los momentos difíciles del camino. Las alegrías, las tristezas, los viajes, los desayunos y las mil y una pláticas fueron apoyo invaluable en esta aventura.

A los diversos directores, personal docente y administrativo del Tecnológico de Culiacán que estuvieron al pendiente de mis avances. En especial a los Dres. Lucía Barrón y Ramon Zataraín por su valioso apoyo y mejores consejos que me permitieron tomar las mejores decisiones en momentos cruciales de esta aventura.

A toda mi familia que siempre creyó que llegaría a la meta, aún cuando a veces no veía luz: a mi mamá Mireya y mi hermano Claudio por sus oraciones y amor incondicional; mi mamá Cristina y mi papá Güero, que un día me aconsejaron "irme a Valencia, porque quizá eso nunca se repita en mi vida y sería muy importante para mi futuro". Un agradecimiento también a quienes me vieron iniciar la senda y no pudieron verla finalizada, pero que su deseo era verme llegar al final: mi mamá Pita y mi papá Claudio.

Alguien a quien nunca olvidaré, el Profr. Ricardo Quintero Villela. Papá: un día me enseñaste que el camino de la educación era el mejor; yo sé que si hoy estuvieras aquí te sentirías muy orgulloso de tu hijo Pichard. En este trabajo hay cosecha de lo que tu sembraste.

Finalmente, y no de menor importancia, agradezco a Dios por sus infinitas bendiciones. Siempre *YHWH Nissi*. En esta jornada el Divino Maestro estuvo presente y en los momentos más cruciales su Palabra fue "*lámpara a mis pies y lumbrera a mi camino...*"

Resumen

La capacidad de comunicación e integración que ofrece Internet, está facilitando el establecimiento y automatización de relaciones entre los participantes de los nuevos Modelos de Negocio electrónico. Aunque no existe una clasificación definitiva de éstos, es posible distinguir la integración de datos y funcionalidad como un requisito fundamental para la implementación de los mismos. Algunos de los casos más significativos lo constituyen empresas como Google y Amazon que ofrecen a sus socios sus plataformas tecnológicas a través de interfaces (APIs) basadas en servicios Web. Estas compañías establecen contratos de negocio que brindan la posibilidad de integración de sus plataformas con las aplicaciones Web de los socios. Así, los diversos servicios que ofrecen les están disponibles y establecen con ello una relación que redunda en beneficio mutuo.

Tres actores básicos se pueden distinguir para lograr esta integración: los servicios Web, las aplicaciones Web y los procesos Negocio-a-Negocio. Los primeros, al facilitar la producción y consumo de datos y funcionalidad independientemente de la implementación tecnológica de los sistemas; los segundos, al ser el tipo más dominante de aplicación que se ejecuta en Internet; y los terceros, al definir los pasos necesarios para la realización de los diversos procesos del negocio.

Desde el punto de vista del modelado conceptual, se encuentran diversas propuestas, la mayoría basadas en modelos UML, para la especificación de estos tres actores. No obstante que en múltiples casos es necesaria su participación mutua, se observa que en muchas de estas propuestas se focalizan en sólo alguno y escasamente se encuentran opciones que los incluyan a los tres. Por ejemplo, desde el punto de vista de la Ingeniería Web, se encuentra un número mínimo de métodos que consideran de manera conjunta en sus propuestas los servicios Web y los procesos Negocio-a-Negocio. La mayoría de los métodos se centran en aplicaciones Web que ofrecen un espacio de navegación sobre grandes colecciones de datos, pero no permiten la integración de datos y funcionalidad de aplicaciones externas, como tampoco la exposición de su funcionalidad para consumo de otras. Por otro lado, desde el punto de los servicios Web, la mayor parte de las aproximaciones considera su especificación sin considerar su inclusión en aplicaciones Web o el papel que juega la interacción humana.

La presente tesis ofrece una propuesta para el modelado conceptual de aplicaciones Web que integran datos y funcionalidad a partir de servicios Web, con consideraciones adicionales a los procesos Negocio-a-Negocio. La solución se expone en el contexto del método de Ingeniería Web *Object Oriented Web Solutions* (OOWS). Este método es la extensión para el modelado conceptual de aplicaciones Web del método clásico OO-Method. En su concepción original, OOWS ha sido diseñado para la consulta y actualización de datos, pero no para la integración de aplicaciones externas. La propuesta de esta tesis incluye un conjunto de adecuaciones y extensiones a sus primitivas conceptuales para la especificación de este tipo de aplicaciones.

Por otro lado, desde el punto de vista de de servicios Web, también se ofrece un método para su modelado conceptual considerando los aspectos de producción, consumo y composición. Se ofrecen primitivas para la captura de los conceptos esenciales de los servicios Web: servicio, operación, parámetros y valores de retorno; ofreciendo primitivas para los tipos de operaciones básicas, según la especificación WSDL: *one-way*, *request-response*, *notify* y *solicit-response*. Se ofrece también una propuesta para la definición de nueva funcionalidad, mediante el establecimiento de relaciones de agregración y especialización entre servicios Web, enmarcado en el contexto de un marco de trabajo multidimensional para su definición precisa. Todo lo anterior se complementa con la definición de una estrategia de transformación de modelos y de generación automática de código, basada en el marco de trabajo de la Arquitectura Dirigida por Modelos (*Model-Driven Architecture*, MDA) del *Object Management Group* (OMG).

Finalmente, desde el punto de vista de procesos Negocio-a-Negocio, la tesis ofrece una método semi-automático para la obtención de un mapa navegacional OOWS, a partir de un modelo de proceso Negocio-a-Negocio que considera la integración de la aplicación Web con aplicaciones externas y actores humanos.

Resum

La capacitat de comunicació i integració que ofereix Internet, està facilitant l'establiment i automatització de relacions entre els participants dels nous Models de Negoci electrònic. Encara que no existeix una classificació definitiva d'aquests, és possible distingir la integració de dades i funcionalitat com un requisit fonamental per a la implementació dels mateixos. Alguns dels casos més significatius ho constitueixen empreses com Google i Amazon que ofereixen als seus socis les seues plataformes tecnològiques a través d'interfícies (APIs) basades en serveis Web. Aquestes companyies estableixen contractes de negoci que brinden la possibilitat d'integració de les seues plataformes amb les aplicacions Web dels socis. Així, els diversos serveis que ofereixen estan disponibles i estableixen amb això una relació que redunda en benefici mutu.

Tres actors bàsics es poden distingir per a assolir aquesta integració: els serveis Web, les aplicacions Web i els processos Negoci-a-Negoci. Els primers,faciliten la producció i consum de dades i funcionalitat independentment de la implementació tecnològica dels sistemes; els segons, son el tipus més dominant d'aplicació que s'executa en Internet; i els tercers, defineixen els passos necessaris per a la realització dels diversos processos del negoci.

Des del punt de vista del modelatge conceptual, es troben diverses propostes, la majoria basades en models UML, per a l'especificació d'aquests tres actors. A pesar que en múltiples casos és necessària la seua participació mútua, s'observa que en moltes d'aquestes propostes es focalitzen en només algun, i escassament es troben opcions que els incloguen als tres. Per exemple, des del punt de vista de l'Enginyeria Web, es troba un nombre mínim de mètodes que consideren de manera conjunta en les seues propostes els serveis Web i els processos Negoci-a-Negoci. La majoria dels mètodes se centren en aplicacions Web que ofereixen un espai de navegació sobre grans col·leccions de dades, però no permeten la integració de dades i funcionalitat d'aplicacions externes, com tampoc l'exposició de la seua funcionalitat per a consum d'unes altres. D'altra banda, des del punt dels serveis Web, la major part de les aproximacions considera la seua especificació sense considerar la seua inclusió en aplicacions Web o sense considerar el paper que juga la interacció humana.

La present tesi ofereix una proposta per al modelatge conceptual d'aplicacions Web que integren dades i funcionalitat a partir de serveis Web, considerant adicionalment els processos Negoci-a-Negoci. La solució s'exposa en el contexte del mètode d'Enginyeria Web *Object Oriented Web Solutions* (OOWS). Aquest mètode és la extesió per al modelatge conceptual d'aplicacions Web del mètode clàssic OO-Method. En la seua concepció original, OOWS ha estat dissenyat per a la consulta i actualització de dades, però no per a la integració d'aplicacions externes. La proposta d'aquesta tesi inclou un conjunt d'adequacions i extensions de les seues primitives conceptuals per a l'especificació d'aquest tipus d'aplicacions.

D'altra banda, des del punt de vista dels serveis Web, també s'ofereix un mètode per al seu modelatge conceptual considerant els aspectes de producció, consum i

composició. S'ofereixen primitives per a la captura dels conceptes essencials dels serveis Web: servei, operació, paràmetres i valors de tornada; oferint primitives per als tipus d'operacions bàsiques, segons l'especificació WSDL: *one-way*, *request-response*, *notify* i *solicit-response*. S'ofereix també una proposta per a la definició de nova funcionalitat, mitjançant l'establiment de relacions d'agregració i especialització entre serveis Web, emmarcat en el contexte d'un marc multidimensional per a la seua definició precisa. Tot l'anterior es complementa amb la definició d'una estratègia de transformació de models i de generació automàtica de codi, basada en l'Arquitectura Dirigida per Models (*Model-Driven Architecture*, MDA) del *Object Management Group* (OMG).

Finalment, des del punt de vista de processos Negoci-a-Negoci, la tesi ofereix una mètode semi-automàtic per a l'obtenció d'un mapa navegacional OOWS, a partir d'un model de procés Negoci-a-Negoci que considera la integració de l'aplicació Web amb aplicacions externes i actors humans.

Abstract

The communication and integration capacity offered by the Internet is facilitating the establishment and automation of the relationships between the partners of the new electronic Business Models. Although there is no definitive classification of them, it is possible to find both data integration and functionality as a fundamental requirement for its implementation. Some of the most significant cases are Google and Amazon, whom offer their clients their technological platforms through interfaces (APIs) based on Web services. These companies define business contracts in order to enable the integration of its platforms with the Web application of their partners. As a result, their different services are available for them and, in consequence, a reciprocal and beneficial relationship can be established.

Three basic actors are playing a role to achieve this integration level: Web services, Web applications and Business-to-Business processes. The first, enabling the production and consumption of data and functionality, aside from the technological implementation of the systems; the second, being the most dominant application type executing in the Internet; and the third, defining the needed steps to the realization of the different business processes.

From the point of view of conceptual modeling there are different proposals, the majority based on UML models, for the specification of these three actors. Although in many situations their mutual participation is needed, it can be observed that many of these proposals are focused on only one role and rarely include the other two. For example, from the Web Engineering standpoint, a minimum number of methods can be found that consider both Web services and Business-to-Business processes in their approaches. The great majority of methods only consider Web applications which offer a navigational space defined over great data collections, but do not allow to integrate data and functionality from external applications, as well as they are not able to expose their functionality for consumption to others. On the other hand, from the point of view of Web services, the majority of approaches only consider their own specification without considering their inclusion in Web applications or the role the human interaction plays.

This thesis offers a proposal for the conceptual modeling of Web applications that integrate data and functionality from Web services, with additional considerations to Business-to-Business processes. The solution is shown in the context of the Web Engineering method *Object Oriented Web Solutions* (OOWS). This method is the extension of the conceptual modeling of Web applications of the classic method OO-Method. In their original definition, OOWS were designed for the consult and update of data, without considering the integration of external applications. The proposal of this thesis includes a set of adaptations and extensions to their conceptual primitives for the specification of this type of applications.

On the other hand, from the point of view of Web services, a method for their conceptual modeling is also offered considering their production, consumption and composition aspects. Primitives for capturing the essential concepts of the Web

services are offered: service, operation, parameters and return values; offering primitives for the basic operation types: *one-way*, *request-response*, *notify* and *solicit-response*. A proposal is also offered for the definition of new functionality through the establishment of aggregation and specialization relationships between Web services, in the context of a multidimensional framework for its precise definition. All is complemented with the definition of a strategy for model transformation and automatic code generation based on the Model-Driven Architecture (MDA) framework of the Object Management Group (OMG).

Finally, from the point of view of Business-to-Business processes, this thesis offers a semi-automatic method to obtain an OOWS Navigational Map, from a Business-to-Business process model that considers the integration of the Web application with external applications and human actors.

Índice

Lista de Figuras

Lista de Tablas

CAPÍTULO 1

Introducción

El surgimiento de la Internet ha traído cambios importantes en la forma en la que las empresas operan sus negocios. Las facilidades de intercambio de información que ésta ofrece, ha llevado consigo el surgimiento de Modelos de Negocio diferentes a los tradicionales. La variedad que poseen estos nuevos Modelos de Negocio, ha dificultado la definición de una taxonomía definitiva de los mismos, aunque algunas clasificaciones se han realizado ([49, 61, 62, 63]). Dentro de las categorías propuestas es posible identificar (con diferentes nombres) un Modelo de Negocio *Negocio-a-Negocio* que se caracteriza por la integración de datos y funcionalidad, así como también por el intercambio y procesamiento electrónico de información entre los socios del negocio. La automatización de este tipo de Modelo de Negocio se apoya en aplicaciones que requieren –igualmente- de tecnologías que permitan la integración de datos y funcionalidad, así como el intercambio y procesamiento de información a través de Internet. La presente tesis se enfoca a este tipo de aplicaciones.

El contexto de esta tesis está enmarcado dentro del ámbito del modelado conceptual de los *servicios Web*, las *aplicaciones Web*, y los *procesos Negocio-a-Negocio*[1]: tres actores fundamentales para el desarrollo de la tecnología que da soporte a los Modelos de Negocio mencionados. Las aplicaciones Web a las que se refiere este trabajo, permiten no sólo la consulta y actualización de datos locales

[1] Un tipo de proceso caracterizado por transacciones que requieren interacción entre dos o más negocios. Son llamados en inglés *Business-to-Business* (B2B)

sino también la producción y consumo de funcionalidad de otras aplicaciones, para lo cual el concepto de *servicio Web* juega un papel fundamental [64].

Las contribuciones de esta tesis parten de un estudio inicial del estado del arte en la *Ingeniería Web*, los *servicios Web* y los *Procesos Negocio-a-Negocio*, detectando áreas de oportunidad para su integración y complemento.

En este contexto, se ofrecen extensiones y adecuaciones al método de *Ingeniería Web* OOWS[9], con el cual se ha tenido, particularmente, experiencia a lo largo de la investigación. Agregando una fase más de *Modelado de Integración* en su método e incluyendo un *Modelo de Servicios* y un *Modelo Dinámico de Composición de Servicios*, para la especificación de la funcionalidad que produce y consume la aplicación. Estos modelos, a su vez, son integrados con los modelos conceptuales originales del método, mediante algunas adecuaciones y extensiones a sus primitivas conceptuales. Con esto se ofrece una solución de modelado conceptual para el tipo de aplicaciones Web mencionadas.

Por otro lado, en el ámbito de los *servicios Web*, otra contribución que ofrecen los nuevos modelos propuestos, es un método que permite la composición y especialización de servicios mediante relaciones de agregación y herencia, acompañado de un marco de trabajo para su definición precisa; complementado con un conjunto de estrategias de transformación de modelos, basado en MDA [36], para la generación automática de código.

Finalmente, en el ámbito de los *Procesos Negocio-a-Negocio*, se ofrece un método semi-automático para la obtención de espacios navegacionales, para el método OOWS.

Este capítulo inicial ofrece una panorámica general de la tesis. Se ha organizado de la siguiente manera: en el primer apartado se presenta la motivación fundamental de este trabajo de investigación. En el segundo apartado se plantea el problema y se ofrecen los puntos centrales de la solución que se desarrolla en los capítulos subsecuentes. En este contexto, el tercer apartado ofrece los objetivos del trabajo; y finalmente, en el cuarto se ofrece una descripción breve de cada uno de los capítulos y apéndices de la tesis.

1.1 Motivación

Hasta la aparición de los *servicios Web*, la producción y consumo de funcionalidad se había implementado mediante diversas tecnologías. Esta diversidad trajo consigo el problema de interoperabilidad: aunque el intercambio de datos y procesamiento fue posible en contextos tecnológicos homogéneos (por ejemplo en el Intercambio Electrónico de Datos, *EDI*[27]), éste se dificultaba en contextos heterogéneos. Esta limitante es importante, ya que para lograr la implementación exitosa del Modelo de Negocio mencionado (integrando datos y funcionalidad, así como procesos *Negocio-a-Negocio*), se requiere facilitar la integración, a pesar de que las empresas no se soporten con tecnologías comunes. Hoy en día, los servicios Web surgen como la solución tecnológica fundamental para lograr este objetivo.

Por otro lado, el reconocimiento de la comunidad de Ingeniería de Software de que las *aplicaciones Web* incluyen aspectos (como la navegación y la presentación) no presentes en las aplicaciones convencionales [2] trajo consigo el surgimiento de la *Ingeniería Web* [1]: una disciplina que propone extensiones y adecuaciones a los métodos y modelos tradicionales, para el modelado conceptual de este tipo de aplicaciones. Algunos métodos representativos de la disciplina incluyen a: OOHDM[4,41], UWE[8], WSDM[66], WebML[12] OO-H[7] y OOWS[9]. El principio subyacente en todos ellos es que una aplicación Web debe desarrollarse partiendo de una descripción precisa en la forma de un Esquema Conceptual (EC). Luego, este EC se transforma a una representación software, mediante un conjunto de correspondencias entre las abstracciones conceptuales que constituyen su EC y componentes software. En particular, la presente tesis se centra en el método OOWS.

Aún cuando la participación de los servicios Web en las aplicaciones Web es cada vez mayor, muy pocos métodos de Ingeniería Web están considerando su inclusión en sus métodos y modelos[2]. En este contexto, el método OOWS no es la excepción. OOWS nace como una extensión para el modelado conceptual de aplicaciones Web a partir de un EC del método clásico de Ingeniería de Software OO-Method [10]. Esta extensión incluye los aspectos de presentación y navegación, pero

[2] WebML [13,62] es uno de los pocos métodos de Ingeniería Web que ha sido extendido incluyendo primitivas conceptuales para el manejo de servicios Web, como parte de su modelo navegacional.

no servicios Web, con lo cual no es posible que la aplicación pueda integrar datos y funcionalidad a partir de servicios Web o ser partícipe en procesos *Negocio-a-Negocio*.

Ante el escenario planteado, esta tesis propone un acercamiento entre la Ingeniería Web y los servicios Web en beneficio de ambas disciplinas, fundamentado principalmente en la introducción de los servicios Web como *ciudadanos de primera clase* en los métodos de Ingeniería Web.

En particular este acercamiento se presenta como una extensión al método de Ingeniería Web OOWS, proporcionándole:

- Un *Modelo de Servicios* y un *Modelo Dinámico de Composición de Servicios*, para la *especificación homogénea de funcionalidad producida y consumida* mediante servicios Web, por la aplicación Web.
- Mecanismos en *el Modelo de Servicios, para la descripción e integración de datos y funcionalidad* en la aplicación Web.
- Mecanismos en *el Modelo de Servicios, para la composición de operaciones* a partir de otros servicios Web; incluyendo relaciones de agregación y especialización entre servicios.
- *Extensiones a las primitivas conceptuales actuales* de Modelado de Navegación, para su integración con el Modelo de Servicios.
- *Extensiones a los patrones de presentación* del Modelo de Presentación, para la especificación de estos requisitos, en el contexto de los servicios Web.
- Un *conjunto de escenarios de transformación de modelos*, para la generación automática de código a partir de los nuevos modelos propuestos.

Los requisitos que la extensión busca satisfacer, se orientan hacia los aspectos de *modelado* y *metodológicos* de la siguiente manera:

- Desde el punto de vista de *modelado*:
  - Minimizar el número de nuevas abstracciones introducidas.
  - Mantener el principio de separación de aspectos.
  - Facilitar la integración con los modelos existentes.
  - Facilitar la generación automática de código.
- Desde el punto de vista *metodológico*:

- Definir guías metodológicas para la construcción de aplicaciones Web basadas en servicios Web.
- Obtener una arquitectura basada en servicios Web para la aplicación Web.

Este tipo de extensiones permitirán al método OOWS la especificación de aplicaciones Web que integran datos y funcionalidad a partir de servicios Web, así como tambien la obtención de aplicaciones Web a partir de Procesos *Negocio-a-Negocio*.

1.2 Planteamiento del problema

El problema, en general, se ubica a partir de la evolución paralela de las disciplinas de la *Ingeniería Web* y la *Orientación a Servicios*:

La mayoría de los métodos de *Ingeniería Web* se han limitado a la especificación conceptual de aplicaciones Web sin considerar la integración con otras aplicaciones, ni el papel que juegan los procesos *Negocio-a-Negocio*.

A su vez, las propuestas actuales de *Orientación a Servicios* y los procesos *Negocio-a-Negocio* no han dado importancia al papel, que en puntos críticos del proceso, debe tomar el humano; ya que la mayor parte de las soluciones se ofrecen considerando la automatización completa, sin su intervención. Considerando que la plataforma tecnológica dominante actual, en la que se realizan estos procesos, es Internet y dado que de forma creciente la interacción humana se está dando a través de interfaces Web, resulta conveniente su acercamiento o aproximación.

La presente tesis propone una aproximación entre ambas disciplinas. En ésta, se detectan los problemas específicos que a continuación se mencionan, cuya solución son, a su vez, áreas de oportunidad que posibilitarían el enriquecimiento benéfico mutuo.

1.2.1 Problemas específicos

Los problemas específicos, que se identifican al acercar la *Ingeniería Web* y la *Orientación a Servicios*, son los siguientes:

- Los métodos actuales de Ingeniería Web están mayormente enfocados a la especificación de vistas hipermediales que consultan y actualizan un do-

minio local relacional u objetual. La integración de datos y funcionalidad externas a la aplicación Web, es poco considerada.

- La especificación de los procesos *Negocio-a-Negocio* se realiza dando mayor énfasis a la colaboración entre aplicaciones y menor énfasis a la participación humana. Ésta última es indispensable en ciertos puntos del proceso y debería considerase también en la definición de los mismos. No se han considerado las prácticas y principios que ofrece la Ingeniería Web, que resultarían útiles para lo mismo.
- El papel que están jugando los procesos de negocio, en los métodos de Ingeniería Web, se ha circunscrito a la obtención de mapas navegacionales en dos modalidades principales: (1) *híbridos*, donde son mezcladas abstracciones navegacionales y de procesos (el principio de "separación de aspectos" es aplicado débilmente); y (2) *puros*, donde el mapa navegacional se obtiene a partir del conocimiento capturado en los procesos de negocio. En cualquier caso, el papel que están jugando los procesos de negocio se limita al desarrollo de aplicaciones Web sobre la funcionalidad interna. Se requieren, también, soluciones que permitan que las aplicaciones Web colaboren con funcionalidades externas.
- Aquellos métodos de Ingeniería Web que consideran la incorporación de los servicios Web como parte de las aplicaciones, no poseen modelos diferenciados que especifiquen sus aspectos internos mediante composición de servicios.
- La composición de servicios Web es un mecanismo importante que se requiere incluir en la especificación de aplicaciones Web que integran datos y funcionalidad. En aquellos métodos de Ingeniería Web que la consideran, suelen definirla como parte (mezclada) de la navegación. Se requieren mecanismos conceptuales para su especificación, así como también mecanismos que permitan su integración con los modelos restantes del método.

1.2.2 Solución propuesta

Los problemas antes detectados podrían resolverse de la siguiente manera:

- Ofreciendo extensiones a los métodos de Ingeniería Web a través de la creación y/o extensión de sus modelos para la captura de los nuevos requisitos (principalmente al Modelo Navegacional).
- Definiendo un *Modelo de Servicios* que permita capturar y exponer la funcionalidad interna de la aplicación así como también posibilitar la inclusión de la funcionalidad ofrecida por aplicaciones externas.
- Definiendo un *Modelo Dinámico de Composición de Servicios* que permita definir servicios complejos a partir de servicios propios y ajenos a la aplicación.
- Definiendo mecanismos a nivel conceptual para la especificación precisa de la composición de servicios, considerando aspectos estructurales y dinámicos.
- Definiendo un *Modelo de Procesos Negocio-a-Negocio* que permitan especificar procesos distribuidos de negocio en los cuales no solamente se incluya la colaboración entre aplicaciones sino también la colaboración humana.
- Definiendo mecanismos que permitan la integración de los nuevos modelos introducidos a los modelos existentes, principalmente a los Modelos de Navegación y Presentación.
- Definiendo reglas de transformación que permitan generar Modelos navegacionales para aplicaciones Web complejas, a partir de especificaciones de los *Modelos de Servicios* y de *Procesos Negocio-a-Negocio*.
- Definiendo reglas de transformación para la generación automática de código a partir de los modelos propuestos.

1.3 Objetivos

El *objetivo general* de este trabajo de tesis es la incorporación de los servicios Web y los procesos Negocio-a-Negocio, como *ciudadanos de primera clase*, en el método OOWS; para la especificación, desarrollo y generación de aplicaciones Web que integran datos y funcionalidad.

1.3.1 Objetivos específicos

El objetivo general se detalla en los siguientes *objetivos específicos* :

1. Revision del estado del arte en métodos de *Ingeniería Web* y en *Orientación a Servicios*, para detectar áreas de oportunidad que permitan su acercamiento.
2. Definición de un *Modelo de Servicios* que permita describir funcionalidad *propia* de la aplicación actual y funcionalidad *ajena* de aplicaciones externas. El Modelo deberá permitir, además, la definición de funcionalidad compuesta a partir de la funcionalidad propia y ajena (composición).
3. Definición de un *Modelo Dinámico de Composición de Servicios* para la especificación de operaciones de servicios Web compuestos.
4. Definición de mecanismos de composición de servicios Web, a nivel conceptual, que tomen en cuenta los aspectos estructurales de la composición, considerando relaciones de *agregación* y *especialización* entre servicios.
5. Definición de un *Modelo de Proceso Negocio-a-Negocio*, para la descripción de procesos que consideran aplicaciones y actores humanos.
6. Extensión y adecuación de las primitivas conceptuales de los Modelos Navegacional y de Presentación para la integración con los nuevos modelos propuestos.
7. Extensión y adecuación de la Arquitectura Software para incluir una subcapa de componentes software que implementan los servicios Web.
8. Definición de un método que permita obtener aplicaciones Web a partir del *Modelo de Proceso Negocio-a-Negocio*.
9. Definición de reglas de transformación de los modelos propuestos a modelos específicos de plataformas tecnológicas para la generación automática de código.

1.4 Métodología de la investigación

Para el logro de los objetivos planteados la investigación se ha fundamentado en la metodología esquematizada en la Figura 1.1 con las fases y pasos siguientes:

1. **Estudio del estado del arte**: donde se revisa el estado del arte en los *métodos de Ingeniería Web* y la *Orientación a Servicios*. También se incluye una revisión de los *Procesos Negocio-a-Negocio*. Con esto se pretende encontrar las oportunidades de acercamiento entre estas áreas que redunden en su complemento y beneficio.
2. **Redefinición del método OOWS**: aqui la investigación se centra en la redefinición del método OOWS en base a los requisitos detectados en la fase anterior. Esta redefinición se enmarca en el contexto del marco de trabajo MDA, incluyendo los siguientes pasos:

   2.1 Definición/Extensión de Modelos PIM: definición de los metamodelos de *Servicios, Dinámico de Composición de Servicios* y *Procesos Negocio-a-Negocio*. Los nuevos metamodeos requieren a su vez adecuación de los modelos de Navegación y Presentación preexistentes, introduciendo nuevas primitivas conceptuales a los mismos.

   2.2 Definición de Modelos PSM: definición de metamodelos para las plataformas tecnológicas que implementarán los componentes software. Se seleccionaron dos plataformas básicas: Java y WS-BPEL.

   2.3 Definición de Transformaciones de Modelos: diseño de las reglas de correspondencia PIM-a-PIM, PIM-a-PSM y PSM-a-Código para la generación automática de código de los componentes software.
3. **Construcción y prueba de Herramienta**: donde los metamodelos y las transformaciones se implementan. Los primeros basados en el marco de trabajo *Eclipse Modeling Framework* (EMF) y las segundas mediante el lenguaje *Query/View/Transformation* (QVT) operacional de Borland Together, para las transformaciones Modelo-a-Modelo; y MOFScript para las transformaciones Modelo-a-Texto.

4. **Escritura de la Tesis**: a la par de las fases anteriores se escribirá la tesis incluyendo sus diversos apartados (capítulos, conclusiones, trabajos futuros y apéndices).

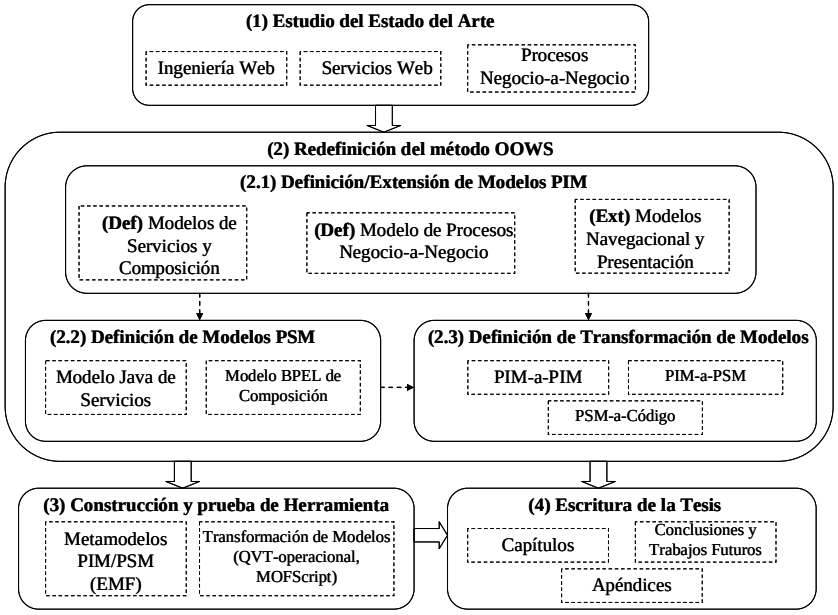


Figura 1.1 Metodología de la investigación

1.5 Estructura de la tesis

La estructura de esta tesis se ha organizado de la siguiente manera:

- **Capítulo 1**. El capítulo actual, donde se introduce y motiva el tema. En este capítulo se define el problema general y sus problemas específicos. Ofrece un conjunto de soluciones que se desarrollan al detalle en los capítulos siguientes de la tesis, así como los objetivos general y específicos de la misma.
- **Capítulo 2**. En el cual se revisa el estado del arte de los métodos de Ingeniería Web que incluyen aspectos de integración en sus técnicas y modelos, a partir de una selección representativa de los mismos. También se revisan las aproximaciones tecnológicas (previas y actuales) para la implementación de dicha integración, dando especial énfasis a los servicios Web y su composición. El objetivo de este capítulo es identificar las áreas de oportunidad que dan origen a la aportación de esta tesis.

- **Capítulo 3**. Una vez revisado el estado del arte, este capítulo introduce el método OOWS. Explica sus modelos y enfoque metodológico actual y ofrece el marco de trabajo general para su extensión, a fin de que sea posible la especificación y generación de aplicaciones Web basadas en servicios.
- **Capítulo 4**. La extensión necesaria para el método OOWS incluye la definición de un *Modelo de Servicios*. Se presenta el estado del arte de algunas aproximaciones típicas para el modelado conceptual de servicios (no realizadas en métodos de Ingeniería Web) y la forma como éstas han sido utilizadas para la definición del modelado de servicios en OOWS. Incluye también la definición del *Modelo Dinámico de Composición de Servicios*, que se utiliza para la definición de operaciones compuestas. Este capítulo explica los elementos de modelado de su metamodelo, con sus propiedades y restricciones. Ofrece también algunos ejemplos de su uso.
- **Capítulo 5**. Una vez definidos los Modelos de Servicios y Dinámico de Composición de Servicios, este capítulo ofrece las adecuaciones y extensiones al modelado navegacional y de presentación de OOWS, así como su integración con los modelos mencionados para la especificación de aplicaciones Web basadas en servicios. Se introduce la primitiva *Unidad Interacción*, para la consulta de datos e invocación de funcionalidad propia y ajena. Se agregan algunas relaciones contextuales más y se extiende el Modelo de Presentación, con patrones específicos para los nuevos requisitos planteados por la presencia de los servicios Web.
- **Capítulo 6**. Ofrece un método para la obtención del Modelo Navegacional OOWS a partir de un Modelo de Proceso Negocio-a-Negocio. El método permite ir desde la especificación del proceso de negocio hasta la obtención de un Mapa Navegacional prototipo.
- **Capítulo 7**. A partir de los modelos definidos en los capítulos anteriores, este capítulo se centra en la implementación de las reglas de transformación *Modelo-A-Modelo* y *Modelo-A-Texto* para la generación automática de código para los modelos propuestos. Se basa en al marco de trabajo de la Arquitectura Dirigida por Modelos (*Model-*

driven Architecture, MDA) y se enfoca en los Modelos de Servicios y Dinámico de Composición de Servicios. La generación de código se organiza en cuatro escenarios.

- **Capítulo 8**. Ofrece conclusiones, trabajo futuro y publicaciones producto de este trabajo de investigación.
- **Apéndice 1**: Incluye un ejemplo completo de implementación del escenario 1: *Importación de Servicios Ajenos*; con metamodelos, modelos, reglas de transformación y código final. El ejemplo se basa en el servicio de búsquedas de `Google`.
- **Apéndice 2**: Incluye un ejemplo completo de implementación del escenario 2: *Generación de servicios propios vistas de clases de negocio*, con metamodelos, modelos, reglas de transformación y código final. El ejemplo se basa en la creación de un servicio propio para una clase de negocio `Producto`.
- **Apéndice 3**: Incluye un ejemplo completo de implementación del escenario 3: *Agregación y Composición de Servicios*, con metamodelos, modelos, reglas de transformación y código final. El ejemplo se basa en el servicio `BestStoreService` que se introduce en el capítulo 4.
- **Apéndice 4**: Incluye un ejemplo completo de implementación del escenario 4: *Especialización de Servicios*; con metamodelos, modelos, reglas de transformación y código final. El ejemplo se basa en el servicio `TravelService` que se introduce en el capítulo 4.
- **Apéndice 5**: Ofrece, finalmente, una breve explicación de los dos lenguajes de transformación de modelos utilizados para implementar los cuatro escenarios de transformación de modelos y generación de código: *QVT* y *MOFScript*.

CAPÍTULO 2

Estado del arte

El siguiente capítulo explora las áreas tecnológicas en las cuales se enmarca el trabajo de la tesis. Se estudian los campos de la Ingeniería Web, los procesos Negocio-a-Negocio y los servicios Web. En cada una de estas áreas se discute su origen, características y tecnologías principales que las constituyen. También se identifican problemas que crean áreas de oportunidad para el acercamiento de las disciplinas, en lo cual se fundamenta la aportación principal de este trabajo de tesis.

El capítulo está organizado en dos apartados principales: en el primero, se exploran algunos métodos de Ingeniería Web que se han considerado representativos porque tratan aspectos relativos a la integración, consumo y producción de funcionalidad externa –principalmente en el campo de los servicios Web. En el segundo, se explican las aproximaciones más importantes en el campo de los procesos Negocio-a-Negocio y los servicios Web. El capítulo ofrece una panorámica general de estas dos áreas, enfatizando también sus fortalezas y debilidades.

2.1 La Ingeniería Web

El nacimiento de la Ingeniería Web está precedido por la realización de diversos foros, orientados principalmente a la discusión de la naturaleza del desarrollo de las aplicaciones Web y la necesidad de aplicar los principios de la Ingeniería de Software al mismo. Un ejemplo de estos foros se dio en 1998. Pressman [2] reune un grupo *sui-generis* de expertos para discutir el tema: formado por gente con mu-

chos años de experiencia y respaldo en la Ingeniería de Software, así como también por jóvenes con experiencia única en aplicaciones Web. Algunos ofrecieron argumentos que soportaban las similitudes entre ambas áreas y algunos otros ofrecieron argumentos que soportaban las diferencias.

Varios investigadores han distinguido diferencias entre las aplicaciones software comunes y las aplicaciones Web ([1,3,68,69]):

- Poseen ciclos de vida cortos: tan pronto como el sistema está terminado debe considerarse su evolución.
- La evolución es continua, trayendo consigo que muchos decidan hacer de lado las tareas de análisis y diseño, recomendadas por la Ingeniería de Software, sobre todo porque muchas de estas prácticas suponen una volatilidad inferior a la que poseen las aplicaciones Web.
- La gente que utiliza aplicaciones Web es más tolerante a errores.
- Es complicado conocer la demografía de los visitantes de las aplicaciones Web.
- La volatilidad de la interfaz es superior al contenido.
- Los usuarios están potencialmente distribuidos por todo el mundo y no es fácil involucrarlos en el proceso de desarrollo de la aplicación.
- La interfaz posee una riqueza multimedia que no poseen las aplicaciones comunes, lo cual convierte al Web en una combinación de disciplinas que no existe en la definición original de la Ingeniería de Software. Resalta, principalmente, una combinación entre ingeniería y arte.
- Los desarrolladores actuales Web no poseen experiencia en Ingeniería de Software. La mayoría son jóvenes que quizá nunca han escuchado sobre la *crisis del software* de la década de los 60's.
- Existe una presión constante por llevar la mayor parte de las aplicaciones actuales al Web, debido a las oportunidades de negocio que posibilitan este tipo de plataforma.
- Existe un crecimiento continuo de sus requisitos y del contenido de su información, que obligan a las aplicaciones Web a ser diseñadas de tal forma que su escalabilidad y mantenimiento se faciliten.

- Deben proveer dinámicamente información en múltiples formatos (texto, gráficos, imágenes, video).
- Mucho del desarrollo recae en conocimiento y experiencia de programadores individuales (o de grupos pequeños) y sus prácticas suelen ser individuales no estandarizadas.
- Carecen de documentación.
- Carecen de pruebas apropiadas.

Los actuales desarrolladores Web ponen obstáculos para utilizar lo que ellos llaman *prácticas fuera de moda de la Ingeniería de Software* al crear este tipo de aplicaciones. Sin embargo, la experiencia práctica con estas aplicaciones muestra que los resultados obtenidos no han sido satisfactorios.

Una encuesta sobre el desarrollo de aplicaciones Web, realizada por el Consorcio Cutter [67], resalta los siguientes problemas en los proyectos Web:

- En el 84% de los casos los sistemas no satisficieron las necesidades del negocio.
- El 79% de las veces no se cumplió con los tiempos programados.
- El 63% de los proyectos excedieron sus presupuestos.
- En el 53% de las veces los sistemas entregados no poseían la funcionalidad requerida.
- En el 52% de los casos el software entregado fue de pobre calidad.

Este escenario motiva, en 1998, a San Murugesan, Athula Ginige, Yogesh Deshpande y Steve Hansen a establecer una nueva disciplina : la *Ingeniería Web* [68], definiéndola de la siguiente manera:

"*La Ingeniería Web trata con el establecimiento y uso de principios sensatos científicos, de ingeniería y gestión, así como con enfoques disciplinados y sistemáticos, para el desarrollo, instalación y mantenimiento exitoso, de aplicaciones y sistemas de alta calidad, basados en el Web*"

Las prácticas y principios de la Ingeniería Web han sido definidos en torno a *métodos de ingeniería Web*. Tales métodos han surgido como esfuerzos de diversos grupos de investigación alrededor del mundo que, basados principalmente en un

enfoque de modelado conceptual, ofrecen soluciones para la especificación y desarrollo de aplicaciones Web. El tratamiento que han dado al problema ha sido más o menos uniforme. La mayor parte de ellos definen modelos cuyas primitivas permiten la captura de los requisitos y características más relevantes de las aplicaciones Web (contenido, navegación y presentación principalmente).

A continuación se estudia, de forma breve, la forma en la que algunos métodos representativos abordan el problema y en especial lo relativo a la inclusión de los servicios Web y los procesos Negocio-a-Negocio. En este sentido, se pretende identificar el tratamiento que han dado a los mismos para definir un marco de referencia que justifique el porque de la propuesta de esta tesis. La premisa es que si los servicios Web han encontrado en el Web la plataforma adecuada para su existencia, entonces los métodos de Ingeniería Web deberían considerarlos como actores importantes de sus propuestas.

2.1.1 OOHDM

OOHDM (Modelo de Diseño de Hipermedia Orientado a Objetos - *Object Oriented Hypermedia Design Model*) [4, 98], fue creado en 1996 por Daniel Schwabe (Pontifícia Universidade Católica do Rio de Janeiro) y Gustavo Rossi (Universidad Nacional de La Plata). Este método surge como una extensión del método HDM [70](Modelo de Diseño de Hipertexto – *Hypertext Design Model*), creado en 1991 por Franca Garzotto, Paolo Paolini y Daniel Schwabe para la creación de aplicaciones hipermediales incluyendo orientación a objetos.

OOHDM considera que el desarrollo de una aplicación hipermedial se da en un proceso que posee cuatro actividades principales:

1. **Diseño conceptual:** en la cual se construye un *Modelo conceptual* que representa los objetos del dominio, sus relaciones y colaboraciones.
2. **Diseño navegacional:** en la cual se construye un *Modelo Navegacional*, una vista subjetiva definida sobre el Modelo conceptual.
3. **Diseño abstracto de la interfaz:** en la cual se definen los aspectos de presentación de la estructura navegacional de la aplicación. Define la forma en la que los diferentes objetos navegacionales aparecerán, cuales objetos de la interfaz activarán la navegación y funcio-

nalidad de la aplicación y cuales transformaciones de la interfaz tomarán lugar y en que momento.

4. **Implementación:** en la cual el diseñador implementa el diseño.

El tipo original de aplicaciones Web consideradas en este método, no tomaba en cuenta la presencia de los procesos de negocio. En un principio, el método fue propuesto como solución a la carencia de abstracciones que tienen los métodos tradicionales de Ingeniería de Software para la especificación de aplicaciones que engloban la metáfora hipermedial, al consultar y actualizar colecciones de datos en las bases de datos.

Schmid y Rossi [5] extendieron los Modelos conceptual y navegacional para incorporar actividades y entidades, así como nodos contenedores de procesos y primitivas de control de procesos para la especificación del proceso en el método.

Para entender su extensión se analiza el ejemplo que presentan en su trabajo y se discuten los aspectos más importantes de su propuesta. La Figura 2.1 muestra el Modelo Navegacional para una tienda de venta de CD's, modelado de la forma tradicional como lo propone el método.

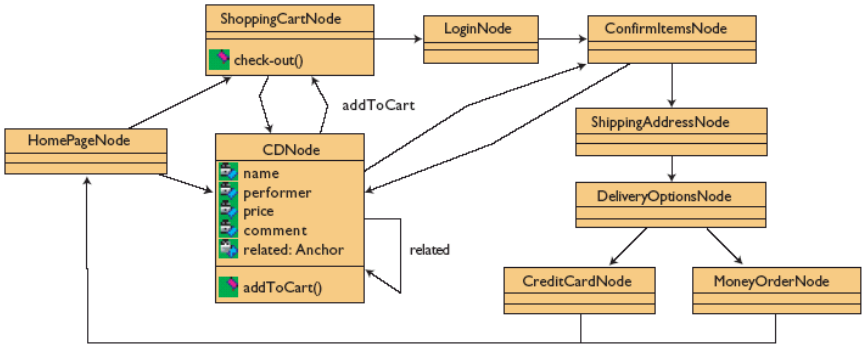


Figura 2.1 Ejemplo de Modelo Navegacional OOHDM

Este modelo incluye (1) *Nodos*, abstracciones de páginas Web que representan vistas de los objetos del dominio y (2) *Enlaces*, que representan rutas de navegación entre los nodos e invocación de operaciones de objetos. Operaciones como *check-out* del nodo *ShoppingCartNode* conllevan a una ruta de navegación que simula el proceso de pago y embarque de un producto: se autentifica el usuario (nodo *LoginNode*), confirma los productos adquiridos (nodo *ConfirmItemsNode*), captura la dirección de embarque (*ShippingAddressNode*), selecciona las opciones de entre-

ga del producto (*DeliveryOptionsNode*) y selecciona el método de pago (*CreditCardNode* o *MoneyOrderNode*). Solamente terminadas estas actividades se establece que el proceso ha concluido.

Este ejemplo permite identificar algunos aspectos que podrían mejorarse en la especificación de procesos de negocio:

- Aunque el conjunto de nodos navegacionales representan al proceso de negocio, éste no se define de forma explícita en el modelo. Para el ejemplo mencionado, no está claro el inicio y el final del proceso.
- Existe la posibilidad de definir puntos intermedios de navegación en el proceso, los cuales posibilitan el rompimiento de la secuencia de actividades definidas para el mismo (por ejemplo, los enlaces entre *ConfirmItemsNode* y *CDNode*).
- Al explorar otras páginas el usuario puede crear estados inconsistentes en los procesos. Por ejemplo, al navegar del nodo *ConfirmItemsNode* hacia *CDNode* da al usuario la posibilidad de agregar un CD existente a su cesta ¿Deberá agregarse dos veces a la cesta?.
- Otra fuente de problemas puede ser utilizar el botón de retroceso (*Back*) durante el proceso de negocio, con lo cual surgen varias preguntas, por ejemplo: *¿Deberán deshacerse las acciones realizadas en la página actual?*

Este tipo de problemas, lleva a concluir que los desarrolladores no pueden modelar y representar adecuadamente procesos de negocio usando primitivas hipermediales y semántica de navegación.

Dos características distinguen a los procesos de negocio de la navegación [6]:

- El proceso dirige al usuario a través de sus actividades.
- El proceso mantiene su estado internamente y este puede cambiar en respuesta a las acciones del usuario. Presionar los botones de navegación no afectan el estado del proceso.

La extensión a OODHM se fundamenta en el principio de la división del proceso de negocio en *entidades de negocio* y *procesos de negocio con actividades*. Esto se refleja en los modelos conceptual y de navegación. La Figura 2.2 muestra el Modelo conceptual y el Modelo Navegacional extendidos para el ejemplo mencionado.

La figura muestra (a) el Modelo conceptual y (b) el Modelo Navegacional para el ejemplo discutido. El Modelo conceptual ahora posee objetos entidades (estereotipados con la palabra clave `<<entity>>`) y objetos actividad (estereotipados con la palabra clave `<<activity>>`) que representan procesos. Igualmente el Modelo Navegacional ahora posee nodos navegacionales y nodos de actividad (estereotipados con las palabras clave `<<entity node>>`, `<<activity node>>` y `<<activity node container>>`).

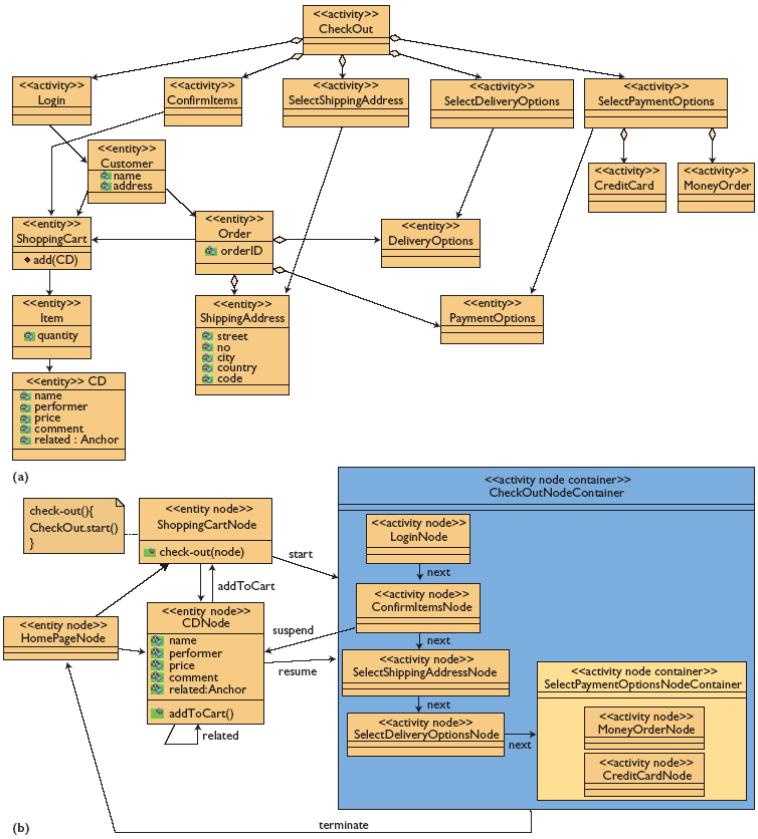


Figura 2.2 Extensión a OODHM para el soporte de procesos de negocio

Aunque no se muestra explícitamente, los objetos entidad, del Modelo conceptual, están asociadas con los nodos navegacionales del Modelo Navegacional; así como los objetos actividad del Modelo conceptual, están asociados a los nodos de actividad del Modelo Navegacional. El proceso de *CheckOut*, que había sido identificado en el Modelo Navegacional de la Figura 2.1, ahora se captura por medio del objeto compuesto actividad *Checkout* del Modelo conceptual y sus actividades com-

ponentes. A su vez, en el Modelo Navegacional se incluye el proceso en el nodo actividad contenedor *CheckOutNodeContainer*.

En el Modelo Navegacional los enlaces entre los nodos de actividad tienen una semántica diferente a los enlaces de navegación: representan el flujo de control entre actividades, no una actividad de exploración en el espacio hipermedial. Los procesos y sus actividades mantienen su estado, de tal manera que el usuario puede cambiar de una tarea de navegación (ej. pasar a explorar el catálogo de CD's y agregarlos a la cesta) a tareas de procesos (ej. pasar al proceso de *Checkout*). Este cambio de tarea posee una semántica para el inicio y terminación del proceso, así como para la suspensión y continuación del mismo.

En esta aproximación se puede destacar lo siguiente:

- La inclusión de los objetos actividad y los nodos actividad en los modelos conceptual y de navegación puede dificultar su comprensión, ya que este par de primitivas no poseen la semántica esperada para estos modelos. Con esta propuesta aún no se resuelve el problema de separación de aspectos.
- Los procesos de negocio son internos. No se considera la integración de funcionalidad de aplicaciones externas.
- No existe la posibilidad de exponer la funcionalidad de la aplicación Web por lo que su integración con otras aplicaciones no es posible. Tampoco existe la posibilidad de integrar datos y funcionalidad externa a partir de servicios Web.

2.1.2 OO-H

OO-H [7,42] (llamado también *OO-HMethod*) fue desarrollado de manera conjunta, en el año 2000, por Jaime Gómez y Cristina Cachero de la Universidad de Alicante y Oscar Pastor de la Universidad Politécnica de Valencia. OO-H "*es un modelo genérico que ofrece al diseñador la semántica y notación necesaria para el desarrollo de aplicaciones Web así como su conexión con la lógica de la aplicación existente*"[7]. La lógica mencionada corresponde a aquella que ofrece el método OO-Method [10].

OO-H propone para el modelado de una aplicación Web el siguiente conjunto de notaciones y técnicas:

- **Un Proceso de diseño:** el cual define las fases que un diseñador debería de cubrir para construir una interfaz Web que satisfaga los requisitos del usuario. Este proceso de diseño parte de un diagrama de clases UML que captura la información estructural del dominio, a partir del cual se define un conjunto de *Diagramas de Acceso Navegacional* (NAD) para cada tipo de usuario, incluyendo la información, servicios y rutas de navegación requeridos para la satisfacción de sus requisitos.
- **Un Modelo Navegacional:** formado mediante un conjunto de uno o más Diagramas de Acceso Navegacional (NAD's).
- **Un Modelo de Presentación:** formado con un *Diagrama Abstracto de Presentación* (APD) el cual captura los requisitos de diseño de presentación para las páginas y su estructura.

Desde su definición original el método ha considerado extensiones adicionales para el modelado de procesos de negocio, las cuales se discuten a continuación.

2.1.2.1 Procesos de negocio en OO-H

Para el modelado de procesos de negocio, OO-H parte de la caracterización de la aplicación Web como *Sistema de negocio* y no solamente como *Sistema de Información*. Como *Sistema de negocio* considera que la aplicación debería estar centrada en objetivos, recursos, reglas y principalmente en el trabajo actual del negocio: sus procesos de negocio.

Su propuesta gira sobre la incorporación de una vista de procesos adicional a las vistas que posee el método. Esta nueva vista se utiliza para definir las vistas restantes (navegación principalmente) de tal forma que la aplicación Web esté diseñada a partir de la definición de la vista de procesos.

El primer paso del método incluye un modelo de análisis que incluye la captura y especificación de requisitos funcionales mediante diagramas de casos de uso UML. La Figura 2.3 muestra el diagrama de casos de uso parcial, para el modelado de la aplicación Web de `Amazon`.

Los casos de uso son de dos tipos, dependiendo del tipo de flujo: (1) *flujo simple*: en el cual el caso de uso comprende solamente actividad de navegación. Este tipo de caso de uso se estereotipa con la palabra clave `<<navigation>>`. (2)

flujo complejo: el cual corresponde a un proceso que se refinará mediante diagramas adicionales. No se estereotipa.

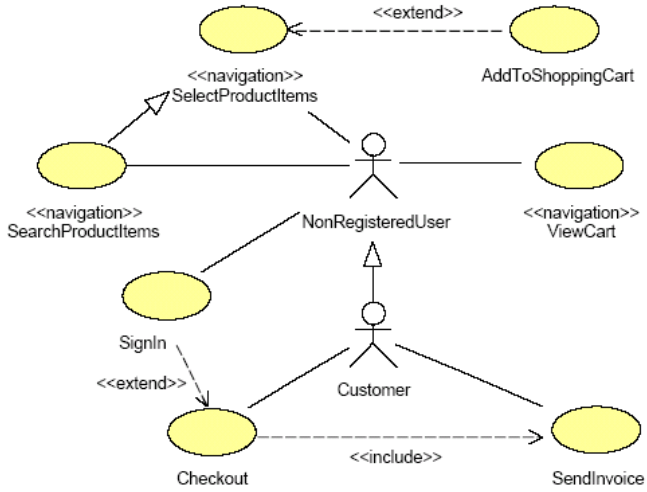


Figura 2.3 Diagrama de casos de uso para `Amazon`

Una vez establecidos los requisitos, el siguiente paso es el análisis del dominio del problema. Este se lleva a cabo mediante la construcción de un modelo conceptual que refleja la estructura del dominio. Para nuestro ejemplo, la Figura 2.4 muestra el Modelo Conceptual para el caso de uso (o proceso) *Checkout*.

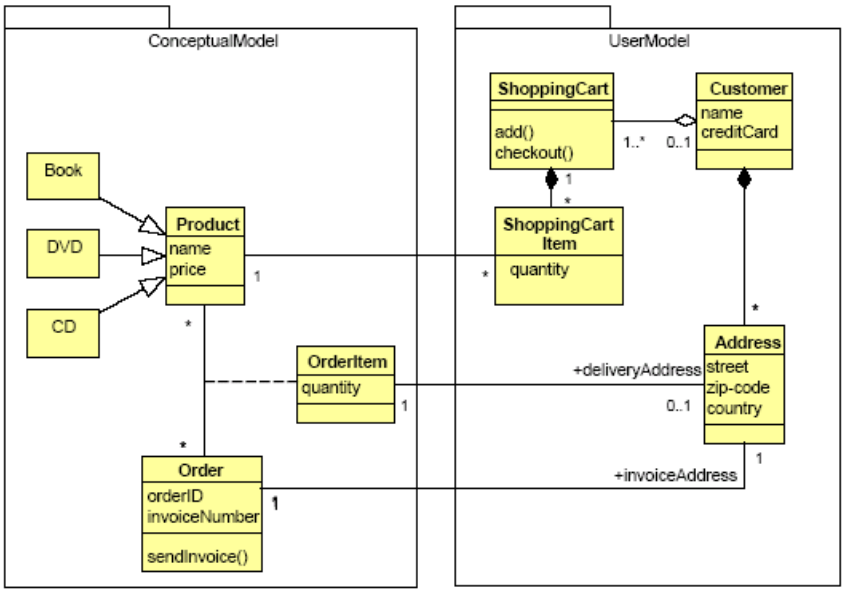


Figura 2.4 El Modelo Conceptual para el proceso de *Checkout*

Este modelo no expresa la vista de procesos del caso de uso *Checkout*, por lo que OO-H agrega un *Modelo de procesos* adicional. Este Modelo de procesos se expresa mediante un *Diagrama de Actividad UML*. A cada caso de uso de flujo complejo, se le asocia uno de estos diagramas para especificar el proceso subyacente. Por ejemplo, la Figura 2.5 muestra el Diagrama de Actividad que complementa la vista estructural del proceso de *Checkout*.

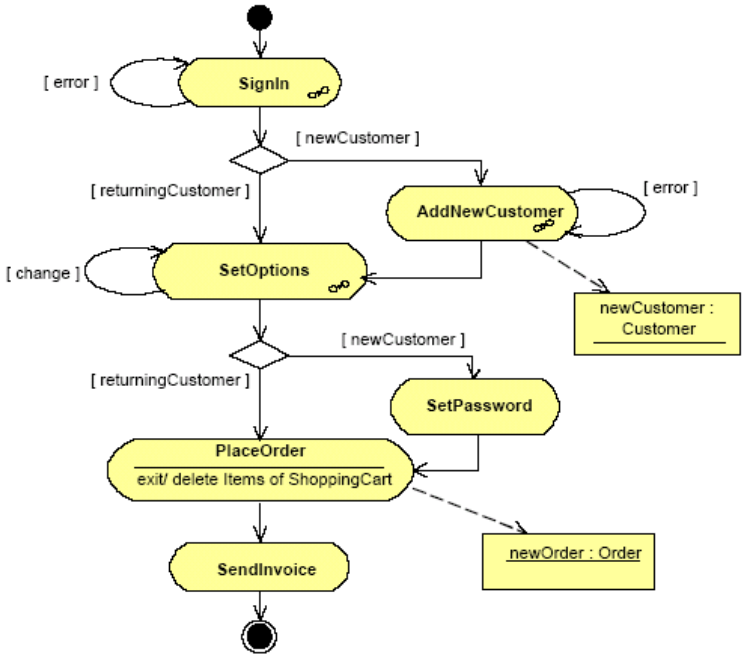


Figura 2.5 El proceso de *Checkout*

Una vez definido este modelo de análisis, OO-H sigue dos posibles enfoques:

- Define el *Modelo Navegacional* a partir del Modelo de proceso.
- Enriquece el *Modelo Navegacional*, agregando en ciertos puntos la capacidad de cambiar de la vista de navegación a la vista de proceso de diseño.

Estos modelos de análisis posteriormente se refinan en modelos de diseño. Por ejemplo la Figura 2.6 es el refinamiento del diagrama de clases de la Figura 2.4 agregando atributos y métodos adicionales.

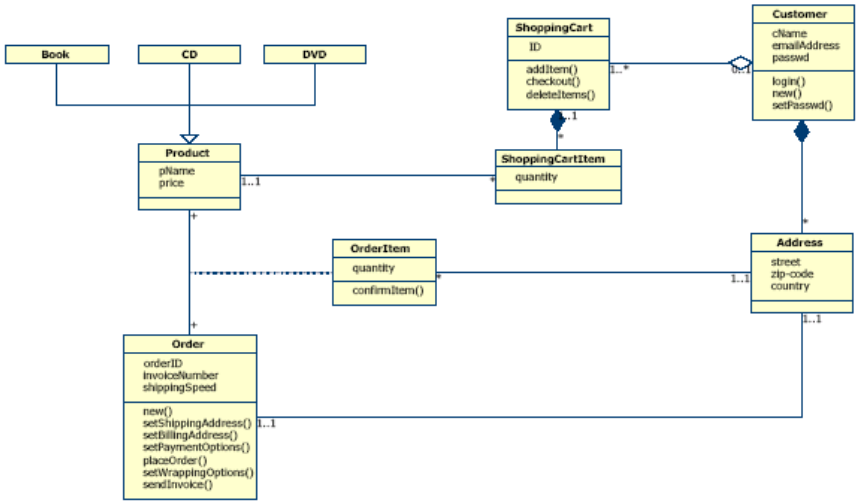


Figura 2.6 Diagrama de clases refinado

Los métodos agregados se utilizan también para el refinamiento del Diagrama de Actividad. Por ejemplo, la Figura 2.7 muestra el refinamiento del Diagrama de Actividad de la Figura 2.5. Algunas de las actividades que estaban como estados de subactividad se realizan mediante estados de *llamada*. Los estados que requieren de soporte transaccional se mezclan en estados de subactividad. A su vez se definen estereotipos para el manejo transaccional (`<<transactional>>` y `<<non-Transactional>>`).

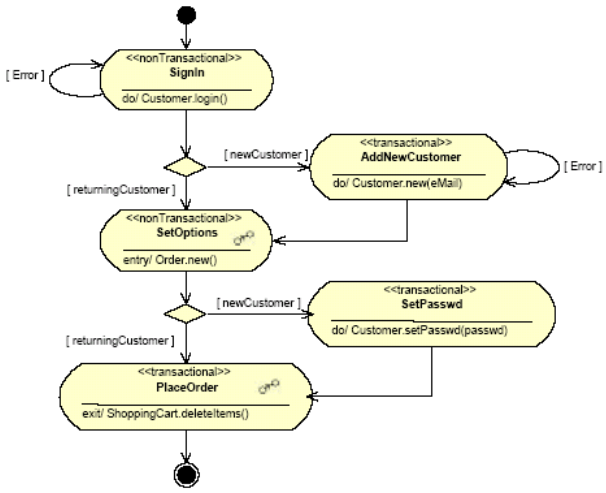


Figura 2.7 Refinamiento del Diagrama de Actividad de *Checkout*

A partir de este Diagrama de Actividad, mediante un conjunto de correspondencias, se obtiene un Modelo Navegacional por defecto. Este Modelo Navegacional se expresa mediante un *Diagrama de Acceso Navegaciona (NAD).* La Figura 2.8 muestra el NAD por defecto que se obtiene para el proceso de *Checkout* .

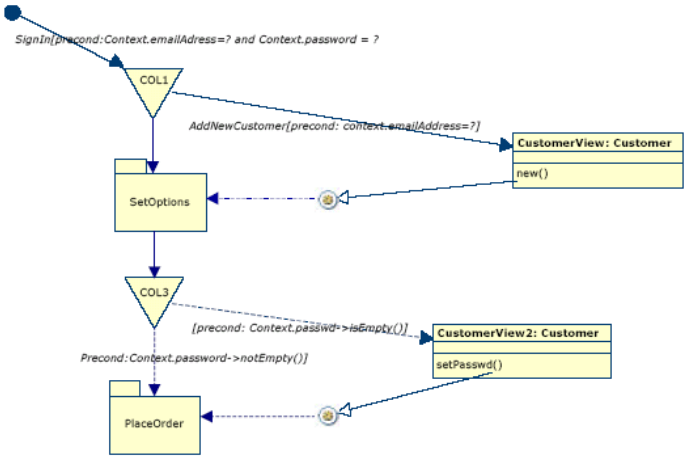


Figura 2.8 NAD por defecto que se obtiene a partir del proceso de *Checkout*

Por ejemplo, la actividad *SigIn* se corresponde con un enlace navegacional que posee un filtro (definido como expresión OCL) que consulta el repositorio de información. Dependiendo del resultado, se forma una colección (`COL1`) que puede dirigir la navegación a la creación de un nuevo cliente (si este no existe) o a un subsistema de navegación *SetOptions*, en el cual el usuario puede formular la orden y colocarla posteriormente para su envío.

Al analizar este enfoque se pueden resaltar los siguientes aspectos:

- Las vistas de procesos y navegación están capturadas en modelos separados, con lo cual se mantiene el principio de separación de aspectos.
- Hay solamente una excepción al punto anterior: cuando el modelo navegacional se enriquece en ciertos puntos en los que la vista de navegación cambia a la vista de procesos. Esto pudiera dificultar su comprensión.
- La vista de procesos sirve como base para la definición del modelo navegacional: el proceso dirige la navegación.

- No existe consumo de funcionalidad de otras aplicaciones. Los procesos se basan en procesos internos de la organización.
- No existe producción de funcionalidad que pueda ser consumida por otras aplicaciones, con lo cual no es posible la colaboración con otras aplicaciones.

2.1.3 UWE

UWE [8] es un método de desarrollo de aplicaciones Web basado en UML estándar. Es desarrollado por Nora Koch, del Instituto de Informática de la Universitat Manchen de Alemania. En UWE el diseño de la aplicación se realiza previo a un proceso semi-automático de generación de la aplicación Web. Basado en un perfil *ligero* UML, sus elementos de metamodelado incluyen un conjunto de primitivas que permiten capturar los requisitos de navegación, presentación, procesos y personalización.

La estrategia de diseño UWE se basa en modelos que se construyen durante la fase de análisis, principalmente el modelo conceptual y el modelo de procesos. A diferencia de OO-H, que traza el modelo de procesos al modelo de navegación, UWE introduce clases específicas de procesos como parte de un modelo separado, que ofrece una interfaz al modelo de navegación.

El diseño de una aplicación Web en UWE sigue los siguientes pasos:

1. Refinamiento del modelo conceptual, agregando atributos y métodos a las clases identificadas.
2. Integración de los procesos al modelo de navegación.
3. Refinamiento del modelo de procesos, construyendo una vista estructural de procesos y una vista del flujo del proceso.
4. Y la construcción de un modelo de presentación basado en los modelos de navegación y procesos, combinando el paradigma navegacional y los procesos de negocio.

2.1.3.1 Los procesos en el modelo de navegación

El modelo de navegación en UWE originalmente se construyó a partir de dos elementos de modelado: la *clase navegacional* y el *enlace navegacional*. A partir de la inclusión de los procesos en el modelo navegacional, se agregan dos elementos más:

- **La Clase de proceso**: cuyas instancias se utilizan por el usuario durante la ejecución del proceso. Cada clase de proceso tiene su correspondencia con un caso de uso no estereotipado con la palabra reservada *navigation*. Cada proceso es refinado posteriormente en el modelo de procesos.
- **El Enlace de proceso**: que modela la asociación entre una *clase navegacional* y una *clase de proceso*. Indica puntos de inicio de un proceso dentro de la estructura navegacional. Pueden ser unidireccionales o bidireccionales. Posee información acerca del estado del proceso mediante una expresión OCL.

La Figura 2.9 muestra un ejemplo de modelo navegacional UWE para la aplicación Web `Amazon`. Cuando una clase navegacional está vinculada con un *enlace de procesos* a una clase de proceso se tiene la posibilidad de navegar y después ejecutar un proceso. Por ejemplo el enlace entre la clase navegacional *ShoppingCart* y la clase de proceso *Checkout* es interpretada como la consulta del carrito de compras y la realización posterior del proceso de *Checkout*. Si el enlace es bidireccional significa que puede realizarse primero la consulta de objetos –a través de la clase navegacional- y después la ejecución del proceso o viceversa. Cada uno de los procesos es definido en un modelo separado.

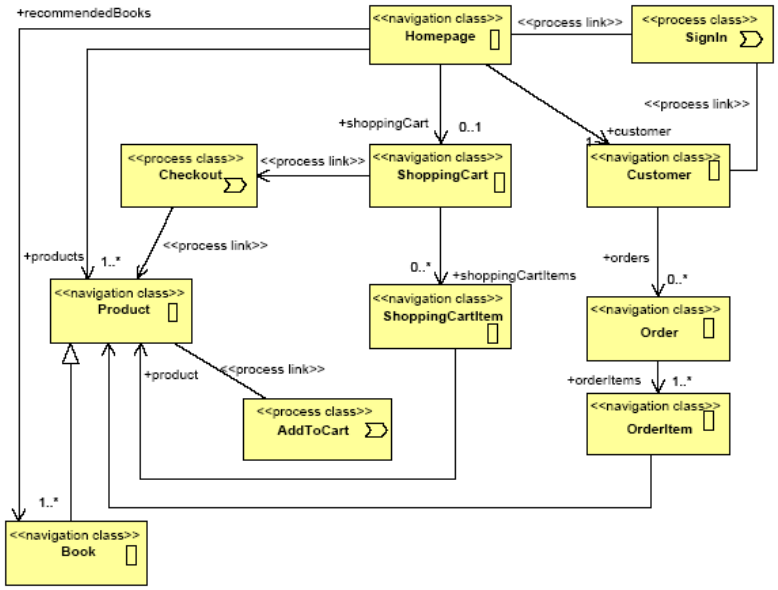


Figura 2.9 Modelo Navegacional UWE para `Amazon`

Este modelo navegacional posteriormente se extiende incluyendo un conjunto de estructuras de acceso necesarias para la navegación: índices, guías turísticas y consultas. Cada una de estas estructuras de acceso están definidas mediante clases estereotipadas `<<index>>`, `<<guided tour>>` y `<<query>>`. Aún más, el modelo se enriquece con menús definidos, de nueva cuenta, con una clase estereotipada `<<menu>>`. La semántica de todas estas construcciones se basa en una extensión del metamodelo de UML con los elementos de modelado específicos de UWE y el uso de OCL para la definición de invariantes.

2.1.3.2 Vistas del modelo de procesos

El modelo de procesos en UWE posee tres vistas: (1) una vista de *integración con el modelo navegacional* –que ya ha sido presentada en el apartado anterior- (2) una *vista estructural* y (3) una *vista de comportamiento* o *modelo de flujo del proceso*. El propósito de las vistas estructural y de flujo del proceso –según UWE- es "*modelar los procesos en sí mismos de forma independiente al modelado de la navegación, buscando con ello la separación de aspectos*" [8].

Por ejemplo, la Figura 2.10 muestra el *Modelo estructural del proceso de Checkout* del caso de estudio. Es un diagrama de clases estereotipado cuyo propósito es capturar la información relacionada con la estructura y comportamiento del proceso.

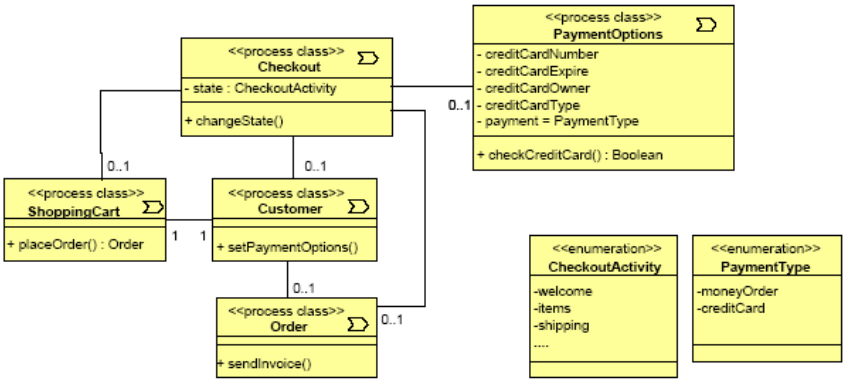


Figura 2.10 El Modelo estructural de procesos

La cardinalidad *0..1* de la clase de procesos *Checkout* y *ShoppingCart* significa que en tiempo de ejecución puede o no existir el carrito de compras asociado al proceso de *Checkout*. Tal como el modelo navegacional se construye a partir del modelo conceptual, este modelo también se construye, en parte, a partir del modelo conceptual. Esto es así porque algunas clases no provienen de este modelo conceptual, por ejemplo la clase de procesos *PaymentOptions* se utiliza para las opciones de pago.

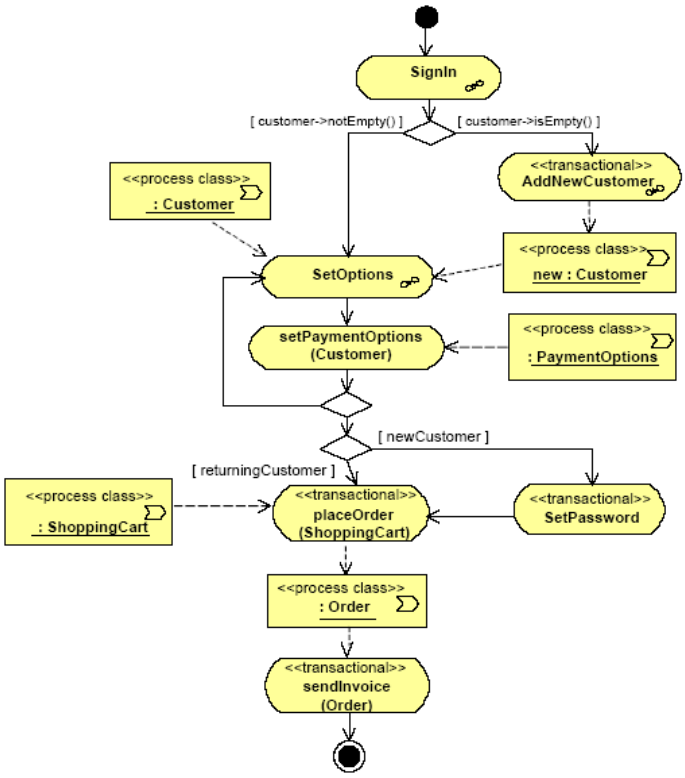


Figura 2.11 El Modelo de flujo de procesos para *Checkout*

Los atributos de este tipo de clases expresan datos necesarios por el proceso, incluyendo la entrada de usuario y el estado de la información del proceso; por ejemplo, mediante la asociación entre la clase *PaymentOptions* y el atributo *state* de la clase *Checkout*.

Cada clase de proceso tiene asociado un *Modelo de flujo de procesos*, expresado como un Diagrama de Actividad UML. Por ejemplo, la Figura 2.11 muestra el Modelo de flujo de procesos para el proceso de *Checkout*. Cada actividad es un estado de *subactividad* o un estado de *llamada* (*call*). En el primer caso representa la

ejecución de un conjunto de pasos con alguna duración, los cuales se modelan en un Diagrama de Actividad separado. En el segundo caso representa la llamada una operación sencilla que previamente ha sido definida en el modelo estructural de procesos. Los *flujos de objeto* (*object flow*) del diagrama, modelan las entradas y salidas del usuario (ver el flujo de objetos entre *PaymentOptions* y *setPaymentOptions*). Los estados de llamada pueden estereotiparse con la palabra clave `<<transactional>>` para expresar el carácter transaccional de este tipo de estados.

En resumen se pueden resaltar los siguientes aspectos de interés para esta tesis:

- La inclusión en el modelo navegacional de la clase de procesos como una primitiva que posee una semántica distinta a los requisitos que se esperan capturar en este modelo podría dificultar la comprensión del mismo.
- No existe integración de funcionalidad disponible en otras aplicaciones. Los procesos se construyen a partir de funcionalidad propia y no se consume funcionalidad ajena disponible en otras aplicaciones.
- No existe tampoco producción de funcionalidad que pueda consumirse por otras aplicaciones, dificultando con ello la definición de procesos Negocio-a-Negocio.

2.1.4 WebML

En sus inicios, WebML fue resultado del proyecto W3I3 (fundado por la Comunidad Europea) enfocado en la creación de “Infraestructura Inteligente de Información” para aplicaciones Web de uso-intenso-de-datos [12]. El principal objetivo de este proyecto (que finalizó el 31 de agosto de 2000) fue proponer una aproximación dirigida por modelos para el diseño de sitios Web.

Actualmente, WebML es desarrollado mediante un trabajo conjunto de la academia y la industria. Desde la academia, por el grupo de investigación en base de datos del Departamento de Electrónica e Información del Politécnico de Milán encabezado por Stefano Ceri, Piero Fraternali, Aldo Bongio, Sara Comai y Maristella Matera. En la industria, soportado por la compañía WebRatio. En los años 2002 al 2004, se inicia el proyecto WebSI, en cuyo contexto WebML y WebRatio se extien-

den para soportar interacción mediante servicios Web y publicación. Actualmente el trabajo de investigación se ha enfocado en el modelado y generación automática de Aplicaciones Ricas en Internet (que utilizan interfaces AJAX o Flash) y en la generación de aplicaciones Web a partir de modelos de procesos.

El método WebML fue diseñado originalmente para la especificación de aplicaciones Web "centradas-en-datos". La especificación conceptual original de una aplicación Web en WebML consistió (y consiste todavía) de un *Modelo de datos* y uno o más *Modelos de hipertexto*. El método está diseñado de tal forma que es posible su extensión. Así, WebML ha incorporado algunos elementos de modelado que permiten la especificación de aplicaciones Web que, además, pueden lograr la interacción con otras aplicaciones mediante servicios Web [13, 95, 96]. A continuación se explican brevemente estos modelos.

2.1.4.1 El Modelo de datos

El Modelo de datos WebML es el modelo Entidad-Relación (ER) de las bases de datos relacionales. Las *entidades* son contenedores de los elementos de datos y las *relaciones* conexiones semánticas entre las entidades. Cada entidad posee un conjunto de *atributos* (con su tipo). Las entidades se pueden organizar en *jerarquías* y las relaciones se pueden restringir mediante *restricciones de cardinalidad*.

2.1.4.2 El Modelo de hipertexto

Sobre el Modelo de datos es posible definir diferentes *vistas del sitio* para los grupos de usuarios (o dispositivos de publicación). Cada vista del sitio es definida mediante un hipertexto (una vista hipertextual), formado mediante un *grafo de páginas* que permite al grupo de usuarios realizar sus actividades específicas.

Las *páginas* consisten de *unidades* conectadas que representan piezas atómicas de información a publicarse. Existen diversos tipos:

- **Unidades de datos**: definidas para una sola entidad, para la cual se define un conjunto de atributos.
- **Unidades multidatos**: que corresponden a todas las instancias de una cierta entidad.

- **Unidades de entrada**: que permiten la especificación de formas para capturar valores de entrada del usuario.
- **Unidades índice**: que presentan múltiples instancias de una entidad como una lista, denotando cada instancia en una entrada de la lista.
- **Unidades *scroller***: que ofrece comandos para realizar exploración a través de todas las instancias de una entidad o de todas las instancias relacionadas a otras instancias.
- **Unidades filtro**: que ofrece campos de entrada para la búsqueda de instancias de una entidad.

La información que se muestra en cada unidad se acompaña con una condición lógica (llamada *selector*) que la determina. Las unidades son conectadas entre sí a través de vínculos que pueden transportar datos de una unidad a otra mediante parámetros. La Figura 2.12 muestra un fragmento de un hipertexto WebML para un par de páginas. Incluye en la página "*Papers page*" un par de unidades –de datos e índice- y en la página "*Authors Page*" una unidad multidatos.

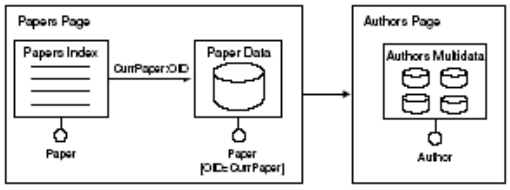


Figura 2.12 Un hipertexto en WebML

No sólo es posible la consulta de los datos sino también su actualización, para lo cual WebML define operaciones de creación, modificación y borrado de instancias de una entidad o la creación y borrado de instancias de una relación.

Los modelos WebML son *extensibles* con lo cual es posible la definición de unidades y operaciones a la medida. Esta característica da origen a las extensiones realizadas para la incorporación de los servicios Web que a continuación se explican.

2.1.4.3 Modelo de datos para Servicios Web

Las extensiones WebML se realizan incluyendo nuevas entidades y relaciones en sus metamodelos de datos e hipertexto. En el caso del metamodelo de datos, la información relativa a los servicios Web (la descripción de sus operaciones, men-

sajes y conversaciones) se incluye dentro del mismo. El metamodelo incluye nuevas entidades y relaciones relativas a la información de los servicios Web.

A continuación se mencionan las unidades para el manejo de servicios Web, extensión realizada sobre la propuesta original del método.

2.1.4.4 Unidades del Modelo de Hipertexto para Servicios Web

El número de unidades del Modelo de Hipertexto ha sido aumentado en seis para incluir las operaciones correspondientes al WSDL [14]. Las dos dimensiones que caracterizan las operaciones son: (1) el *sentido de la operación*: uno o dos sentidos y (2) el *tipo de comunicación*: síncrona o asíncrona. La Figura 2.13 muestra los gráficos utilizados para las primitivas. Las dos últimas corresponden a unidades que indican el inicio y fin de una conversación, indicadas mediante una marca circular en la primera y última operación de la misma.

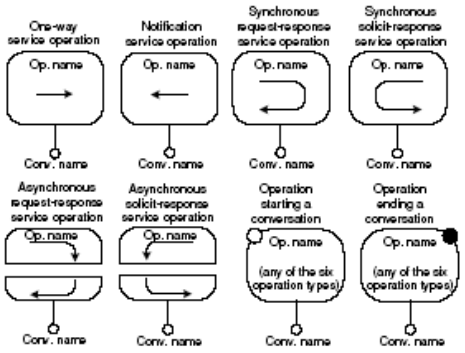


Figura 2.13 Unidades para comunicación con servicios Web

La utilización de estas unidades puede dividirse en dos categorías, explicadas desde el punto de vista de quien las invoca [71]:

Las que son iniciadas por el *cliente*:

1. *Operaciones en un sentido* (*One-way*), que consisten en un mensaje enviado al servicio por el cliente.
2. *Operaciones consulta-respuesta* (*Request-response*), que consisten en un mensaje de consulta enviado por el cliente y un mensaje de respuesta construido por el servicio y enviado al cliente.

Las que son iniciadas por el *servicio*:

1. *Operaciones de notificación* (*Notification*), que consisten en un mensaje de salida del servicio enviado al cliente.
2. *Operaciones de solicitud* (*Solicit*) y *respuesta* (*Response*) creadas para recibir un mensaje de solicitud del cliente y ofrecer un mensaje de respuesta al cliente.

La primer categoría incluye operaciones que se integran dentro de la especificación de la aplicación Web; mientras que la segunda incluye operaciones que no se utilizan en la misma, sino que representan servicios públicos de la aplicación (llamados en conjunto *vista de Servicios*).

2.1.4.5 Integración de las Unidades de Servicios Web en las aplicaciones Web

Utilizando las nuevas unidades de interacción es posible la especificación de aplicaciones Web que pueden invocar operaciones de servicios Web.

La operación en un sentido (*One-way*) es la más simple. Ésta se dispara cuando el usuario navega desde una unidad de entrada hacia una unidad de este tipo, formándose un mensaje a partir de los valores capturados y enviándose al servicio como solicitud.

La operación consulta-respuesta (*Request-response*) modela una operación con un mensaje de entrada, que se envía al servicio y un mensaje de salida, que se recibe desde el servicio. De manera semejante a la operación en un sentido, esta operación se dispara cuando el usuario navega desde una unidad de entrada a una unidad de este tipo. El usuario espera la llegada del mensaje de salida para continuar la navegación a la siguiente unidad.

Existen también algunas aproximaciones para la conceptualización de estas operaciones en modalidad asíncrona [72, 97].

2.1.4.6 Modelando la publicación de Servicios Web

A diferencia del caso anterior, donde la interacción del usuario es necesaria para invocar el servicio Web, en la publicación de servicios Web no es así. Las acciones que se realizan cuando se disparan las operaciones de notificación o solici-

tud-respuesta no se especifican a través de páginas, sino como una cadena de operaciones a realizar sin intervención del usuario. Así, la publicación de servicios Web se especifica de forma separada a la vista del sitio de la aplicación Web. Esta especificación incluye dos conceptos:

- *Vista de servicios*: una colección de puertos, los cuales exponen la funcionalidad de un servicio Web a través de operaciones.
- *Puerto*: que representa el servicio individual, compuesto de un conjunto de operaciones, cada una de las cuales es modelada mediante operaciones solicitud-respuesta y/o notificación.

Una operación se describe como una cadena de operaciones WebML, las cuales especifican las acciones a realizar como consecuencia de la invocación del servicio y posiblemente construya un mensaje de salida que será devuelto al invocador.

2.1.4.7 Papel que juegan las unidades de servicios Web

El papel que juegan las unidades de servicios Web es diverso:

- **Como fuentes de datos:** en el cual los datos devueltos por las operaciones del servicio Web se utilizan para definir el contenido de las páginas. La integración del servicio Web requerirá del manejo de los mensajes de entrada y salida (en formato XML [29]), para lo cual se define una unidad de transformación XML llamada *adaptador*. Esta unidad permite componer el mensaje de entrada del servicio Web; descomponer el mensaje de salida del servicio Web; llamar a múltiples servicios Web en secuencia; en todos los casos adaptando las entradas a las salidas.
- **Para almacenar los datos recuperados de los Servicios Web:** en el cual los datos devueltos por los servicios Web se pueden almacenar en el repositorio local de datos con el fin de volver a ser utilizados una vez que termina la invocación al servicio Web. Existen dos modalidades posibles: (1) *almacenamiento persistente*: con operaciones que actualizan la base de datos y (2) *temporal*: en la cual los datos tienen un tiempo de vida limitado a la sesión del usuario; en la cual los datos son almacenados en memoria principal y no en la base de datos.

- **Creación de nuevas instancias mediante múltiples llamadas de Servicios Web:** en el cual un servicio Web puede llamarse múltiples veces para crear instancias de un mismo sub-esquema de datos. O múltiples servicios se pueden invocar para obtener las distintas piezas de información de un mismo objeto o sub-esquema.

En resumen se pueden resaltar los siguientes aspectos de la propuesta:

- WebML es uno de los pocos métodos que ha abordado el tema de la integración de servicios Web para aplicaciones Web.
- Las nuevas unidades que extienden los Modelos de datos e Hipertexto para la incorporación de servicios Web poseen una semántica diferente a la original de los modelos mencionados. Ahora los modelos capturan aspectos distintos: por ejemplo, el Modelo de Hipertexto no es solo una vista hipertextual de datos o funcionalidad consumida por los servicios Web, sino que la especificación de alguna funcionalidad (en el caso de composición) está dado en el modelo mismo.
- El nivel de abstracción de las nuevas unidades está centrado en tecnologías concretas (XML, servicios Web, etc.). Si bien la motivación de las mismas son los servicios Web, sería conveniente no perder de vista la independencia tecnológica para abordar tecnologías futuras.
- Algunos de los problemas resueltos, sobre todo los relacionados con la persistencia temporal o permanente de los datos devueltos por las operaciones de los servicios Web, no resultan muy claro a que tipo de requisitos satisface. El nivel de abstracción de las soluciones sigue siendo cercano a la tecnología.
- La creación de instancias a partir de los datos de los servicios Web no parece un requisito muy útil para las aplicaciones Web.
- No existe aproximación alguna (a nivel metodológico) hacia la producción de funcionalidad a partir de los modelos existentes.

2.1.5 Resumen comparativo de los métodos de Ingeniería Web

Aunque no han sido discutidos todos los métodos existentes de Ingeniería Web, los métodos seleccionados son un grupo representativo de la disciplina al

momento de la escritura de esta tesis. Con el fin de tener un comparativo a *primera vista* de los mismos (y con ello también una descripción general de la situación actual de todos) se resumen en la Tabla 2.1 los aspectos más relevantes, en cuanto a sus capacidades para el modelado de servicios Web, su inclusión en el método, la producción y consumo de funcionalidad y los procesos de negocio.

Método	**Modelado de Servicios**	**Primitivas para Modelado de Servicios**	**Modelado de Procesos Negocio-a-Negocio**	**Separación de aspectos**	**Producción y consumo de funcionalidad**
OOHDM	No existe	No existen	Implícito en el modelo navegacional. Sólo modela procesos locales	Mezcla de aspectos en el modelo de navegación	No existe
OO-H	No existe	No existen	Sólo modela procesos locales	Los modelos capturan aspectos separados	Se ofrecen algunas soluciones *ad-hoc* y con un nivel de abstracción cercano a la tecnología
UWE	No existe	No existen	Sólo modela procesos locales	Se respeta parcialmente. En algunos modelos hay primitivas con diferente semántica	No existe
WebML	Se modelan los servicios, aunque no en un modelo separado	Posee primitivas en el modelo de hipertexto para los servicios simples y compuestos	Posee un modelo de procesos basado en wfmc	Los modelos poseen primitivas con diferente semántica y se definen a un nivel de abstracción cercano a la tecnología	Si existe

Tabla 2.1 Resumen comparativo de métodos de Ingeniería Web para el modelo de aspectos colaborativos

Como se puede observar, la mayor parte de los métodos no ofrece primitivas para la captura de los aspectos de integración. La excepción es WebML, aunque una mayor aplicación del principio de separación de aspectos y un mayor nivel de abstracción en sus primitivas podrían ofrecerle mejoras.

Algo importante a resaltar es el hecho de que los métodos que consideran el modelo de procesos en sus métodos, suelen hacerlo para procesos de negocio locales, no considerando la integración con aplicaciones externas. Esta capacidad sería

muy conveniente incluirla ya que cada vez son más las aplicaciones de negocio-electrónico (*e-business*) que requieren de dicha capacidad.

No todos los métodos consideran modelos específicos y separados para la producción y consumo de funcionalidad. Esto tiene su explicación en el hecho de que la mayoría considera el modelado de servicios como una extensión de sus modelos existentes. Esto puede disminuir la capacidad de captura y especificación de aspectos más complejos, como la orquestación de servicios.

Después de este análisis se puede concluir que en un entorno cada vez más demandante de aplicaciones negocio-electrónico que faciliten la colaboración entre socios de negocio, la inclusión de los aspectos relativos a los servicios Web es pieza vital para las aplicaciones Web y es poca su participación en los métodos de Ingeniería Web. Su inclusión en los mismos sería conveniente.

2.1.6 Futuras direcciones en la Ingeniería Web

Por otro lado, considerando las tendencias en el uso que se está dando al Web, así como el estado actual de los métodos, es posible visualizar en el futuro a corto y mediano plazo algunas direcciones o líneas de trabajo para la Ingeniería Web:

- El modelado de aplicaciones Web 2.0 y la Web semántica [99,100].
- La definición del papel que puede jugar la Programación Orientada a Aspectos [101] en sus métodos.
- El manejo de la evolución de los sistemas Web.
- El aseguramiento y evaluación de la calidad.

Además, un aspecto de interes que puede impulsar el fortalecimiento de la disciplina es la búsqueda de la interoperabilidad de los métodos.

En este sentido cabe destacar el esfuerzo de la red MDWEnet [102], resultado de talleres realizados en Sydney (2005) y Menlo Park (2006) e iniciada en Diciembre de 2006 en Munich. Esta red incluye a miembros de los grupos UWE, OO-H y WebML así como de las Universidades Málaga, Technical University of Viena y Johannes Kepler University of Linz, que contribuyen con sus conocimientos en el campo de la Ingeniería Web [103, 104]. El alcance de la iniciativa MDWEnet es lograr el desarrollo dirigido por modelos de las aplicaciones Web utilizando diferentes métodos, buscando la interoperabilidad de sus artefactos y modelos.

Su objetivo es mejorar la disciplina haciendo uso conjunto de sus fortalezas, experiencia y conocimiento actual. Para lograr este objetivo se plantean dos caminos básicos: el uso de un metamodelo común o la definición de transformaciones entre los metamodelos existentes. En cualquier caso, parece que este tipo de esfuerzos podría colocar y desarrollar a la Ingeniería Web a un nivel importante a corto y mediano plazo.

2.2 Procesos de negocio y Servicios Web

Además del papel que juegan los servicios Web en los métodos de Ingeniería Web es conveniente una revisión del origen y estado actual de los procesos de negocio, expuestos brevemente a continuación.

2.2.1 Origen de la automatización de los procesos de negocio

Se define un proceso de negocio como una "*colección de actividades realizadas por usuarios humanos o aplicaciones software que de manera conjunta constituyen los diversos pasos para completar un objetivo particular de negocio"* [15].

En un principio, los procesos de negocio se especificaron de manera formal y ejecutable mediante un *flujo de trabajo*[3] [65] diseñado, desarrollado y ejecutado en algún *sistema gestor de flujos de trabajo (SGFT)*. Los SGFT tuvieron su origen en la *automatización de oficinas*, que buscaban la automatización de los procesos administrativos. La idea fundamental fue utilizar documentos electrónicos en lugar de documentos en papel y los SGFT se encargaron de su enrutamiento mediante correo electrónico y formas Web [16]. Conforme la tecnología de *flujos de trabajo* maduró, surgió también la necesidad de la interoperación de aplicaciones, esto es, no solamente controlar el despacho de información entre los participantes humanos, sino también definir la lógica del negocio que permitiera integrar sistemas distribuidos heterogéneos.

El principio fundamental de los *flujos de trabajo* es que actuarían como herramientas integradoras de aplicaciones empresariales, automatizando el control y el flujo de datos entre las aplicaciones. Este aspecto fue también el causante de que al paso del tiempo el interés por estas soluciones se fuera perdiendo ya que dicha

[3] En inglés *workflow*

integración resultó complicada y en muchos de los casos llevó a definiciones pesadas y complejas, poco factibles de implementar en la práctica. La causa principal fue el problema de la interoperabilidad ya que resultaba complicado la colaboración entre aplicaciones implementadas con tecnologías diferentes.

2.2.2 Soluciones al problema de interoperabilidad

Para resolver el problema de la interoperabilidad se requería de una solución que permitiera la interacción de las aplicaciones, independientemente de la plataforma tecnológica en la que fueron implementadas. Este requisito dio origen a la creación de diversas plataformas mediadoras[4] [28] (*RPC* [17], *TP Monitors* [18], *Object brokers* [19], *Object monitors* [20] y *Message brokers* [21]). Cada una de estas plataformas representó un paso evolutivo que hoy se cristaliza en los servicios Web [15]. Gracias a éstos, los conceptos subyacentes a la teoría de *flujos de trabajo* se han vuelto a retomar y se considera que su promesa puede realizarse de forma práctica y efectiva.

La influencia que tiene la teoría de *flujos de trabajo* en los servicios Web se distingue en tres aspectos principales: el *interno*, al definir servicios Web a partir de otros servicios Web (llamado *composición de servicios*); el *externo*, al definir protocolos de coordinación que especifican las *conversaciones* (secuencias de intercambios de mensajes) válidas entre servicios Web; y en la *automatización de los procesos de negocio*.

Los dos primeros aspectos han llevado a considerar que los servicios Web deberían describirse no solamente por su interfaz sino también por el conjunto de conversaciones correctas que pueden tener y por la forma en la que colaboran con otros para el logro de los objetivos del proceso. En este sentido, se están definiendo modelos UML que capturan los aspectos relativos a la conversación [22], los cuales están siendo transformados al Lenguaje de Ejecución de Procesos de Negocio (*Business Process Execution Language- BPEL* [23]).

El tercer aspecto ha llevado a la creación de estándares industriales para la automatización de procesos de negocio. Así, organizaciones como la Iniciativa para la Gestión de Procesos de Negocio (*Business Process Managemente Initiative* -

[4] En inglés *middleware*.

BPMI.org) buscan la convergencia de tecnologías orientadas hacia la computación empresarial, como J2EE y SOAP.

Esta misma organización incluye diversas especificaciones para procesos de negocio: el Lenguaje de Modelado de Procesos de Negocio (*Business Process Modeling Language – BPML*), un metalenguaje para el modelado de procesos de negocio que incluye un modelo de ejecución para procesos Negocio-a-Negocio basado en el concepto de una máquina de estado finito; la Notación para el Modelado de Procesos de Negocio (*Business Process Modeling Notation – BPMN*), una notación gráfica para expresar procesos de negocio la cual se puede vincular con las primitivas de lenguajes de ejecución de procesos, como BPML o BPEL; y el Lenguaje de Consulta de Negocio (*Business Process Query Language – BPQL*), un lenguaje para ser utilizado en Sistemas Gestores de Procesos de Negocio (BPMS) para la consulta de instancias de procesos de negocio.

Otras organizaciones como el Centro de las Naciones Unidas para Facilitar el Comercio y los Negocios Electrónicos (*United Nations Centre for Trade Facilitation and Electronic Business- UN/CEFACT*) y la Organización para el Avance de los Estándares Estructurados de Información (*Organization for the Advancement of Structured Information Standards* - OASIS) de manera conjunta, han definido el Lenguaje XML para Negocios Electrónicos (*ebXML* - electronic business XML) [24]: un conjunto de especificaciones que de forma modular permite conducir negocios sobre Internet en base a intercambio de mensajes XML, conducción de relaciones de negocio, comunicación común de datos y definición y registro de procesos de negocio. ebXML busca aprovechar y suceder a lo ya realizado en el campo del Intercambio Electrónico de Datos (*Electronic Data Interchange - EDI*).

La UN/CEFACT desarrolla también un método de modelado de procesos de negocio e información llamado Metodología de Modelado UN/CEFACT (*UN/CEFACT Modeling Methodology - UMM*[25]). UMM se basa en el estandar ISO 14662 que define el Modelo de Referencia Abierto EDI (*Open EDI*) y que es una especialización del Proceso Unificado de Rational (*Rational Unified Process - RUP* [26]). Ofrece un *marco de trabajo* que de manera uniforme permite representar conceptos, relaciones y semántica para la especificación de procesos a través de un metamodelo UMM. Este metamodelo UMM se define como un perfil UML que posee la semántica requerida para la colaboración de negocios.

2.2.3 Servicios Web

Existen múltiples definiciones de servicio Web, cada una de las cuales enfatiza alguna de las propiedades que posee, de tal manera que en conjunto la mayoría de estas definiciones representan algún aspecto complementario que lo caracteriza. Una definición simple y adecuada para el propósito de la presente tesis es la que proporciona Casati [30]:

"*Un servicio Web es una función de negocio disponible a través de Internet por un proveedor del servicio, accesible por clientes que pueden ser usuarios humanos o aplicaciones software*".

Las interacciones en los servicios Web involucran tres tipos de participantes (ver Figura 2.14): el *productor del servicio*, el *registro del servicio* y el *consumidor del servicio*.

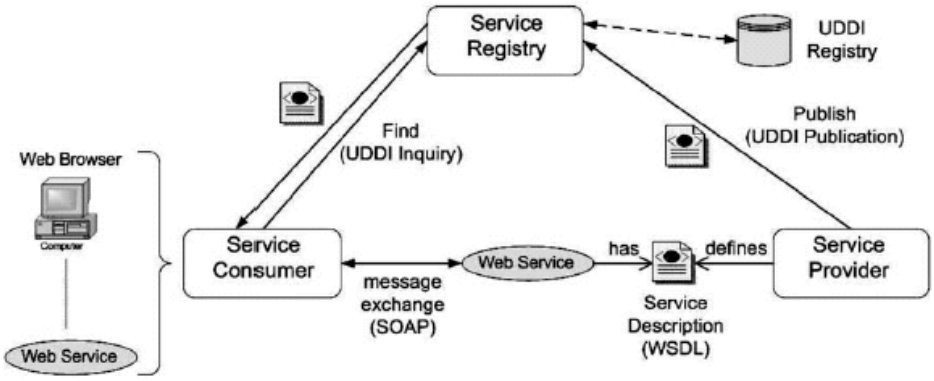


Figura 2.14 Participantes en los Servicios Web

El *productor* del servicio lo ofrece y publica en un *registro* de servicios. Éste es un repositorio de descripciones en el cual se pueden hacer diversas búsquedas. Cada una contiene detalles de los mismos, como sus tipos de datos, operaciones y localizaciones de red. El *consumidor* utiliza la operación de búsqueda (*find*) para localizar servicios de interés, con lo cual el registro recupera la descripción de los servicios relevantes. Con la descripción obtenida, el consumidor invoca el servicio Web correspondiente.

El soporte de las interacciones entre servicios Web está dado por los siguientes estándares tecnológicos:

- **WSDL (Web Services Description Language)** [14]: un lenguaje XML para describir las características operacionales (la interface) de los servicios Web.
- **UDDI (Universal Description, Discovery and Integration)** [31]: que define una interfaz programática para publicación (API de publicación) y descubrimiento (API de consulta) de servicios Web. Su repositorio central es el *registro de negocios*, el cual conceptualmente consiste de *páginas blancas* (información de contacto), *páginas amarillas* (categorización industrial) y *páginas verdes* (información técnica de servicios).
- **SOAP (Simple Object Access Protocol)** [32]: un marco de trabajo (*framework*) ligero de mensajes para el intercambio de datos XML entre servicios Web. Se puede utilizar con varios protocolos de transporte (tales como HTTP, SMTP, FTP).

El uso directo de estos estándares es difícil en la práctica. Una forma en la que se busca facilitar la tarea a los programadores es mediante marcos de trabajo que los entornos de programación suelen ofrecer (ej. Axis[53] en Java). Básicamente el marco de trabajo oculta los detalles de uso de la especificación y permite al programador continuar utilizando las instrucciones y mecanismos que ofrece el lenguaje de programación (con el cual suele estar familiarizado) y evita con ello el uso directo del XML.

Además del marco de trabajo, los entornos de programación incluyen compiladores de *stub's*[5]. Un *stub* es una pieza de código que facilita la comunicación del código del programador con el código del marco de trabajo. Partiendo de la interfaz WSDL, el compilador de *stub's* los genera y el código resultante es el que utiliza directamente el programador en sus aplicaciones.

Si bien los Ambientes de Desarrollo Integrado suelen incluir herramientas que automatizan todo este proceso, es necesario aún que el programador se centre en los detalles técnicos del uso del marco de trabajo y desvíe su atención a lo esencial : la solución del problema del negocio.

[5] Una pieza de código ligero que media entre el código del programador y el código del marco de trabajo.

2.2.4 Composición de servicios Web

Los servicios Web pueden combinarse para ofrecer nuevos servicios llamados *compuestos*. Esta capacidad de combinación es básica para las Arquitecturas Orientadas a Servicios[6] ya que permite la definición servicios cada vez más complejos a partir de otros fundamentales.

En los últimos años la composición de servicios se ha implementado con diversas tecnologías (WSFL[33], XLANG[34], BPML[73], BPEL[35]). De todas ellas, el Lenguaje de Ejecución de Procesos de Negocio (*Business Process Execution Language-BPEL*), propuesto en julio de 2002 por BEA, Microsoft e IBM, representa una convergencia de varias de las características de sus anteriores lenguajes ganando el soporte de muchas compañías (como SAP o Siebel). Con su traspaso a OASIS parece que será en el futuro cercano el estándar *de-facto* para la composición.

En BPEL es posible especificar la composición de forma ejecutable y abstracta. En el primer caso, se define la lógica de implementación interna del servicio compuesto. En el segundo, solamente se define la secuencia de mensajes (de entrada y salida) soportados, es decir su comportamiento externo.

Un servicio compuesto se define como un documento XML que incluye los siguientes aspectos:

- Los diferentes roles de los servicios participantes. Estos servicios colaboran con el servicio compuesto mediante intercambio de mensajes.
- Los tipos de puertos que deben soportar los diferentes servicios componente y el servicio compuesto.
- La definición de la lógica de composición (también llamada orquestación).
- La información de correlación, definiendo como es que los mensajes se pueden enrutar a las instancias de composición correctas.
- La inclusión de variables, con las cuales es posible mantener el estado del servicio compuesto.

Una vez definido el documento XML, es necesario un ambiente de ejecución para el mismo. Este ambiente crea instancias del servicio compuesto y las ejecuta

[6] En inglés *Service Oriented Architecture* (SOA).

invocando a los diversos servicios componente, de acuerdo a la programación del servicio.

Los programadores cuentan también con Ambientes de Desarrollo Integrado para ayudarles en la definición de los servicios compuestos BPEL, sin embargo en todos ellos se requiere que el programador tenga conocimiento al detalle de los diversos elementos XML del programa BPEL. A menos que el programador sea experto en el lenguaje, no le resultará trivial su uso. Otra dificultad adicional la representa el hecho de que ciertas instrucciones BPEL requieren que el programador conozca algunas tecnologías adicionales XML. Por ejemplo, en el caso del manejo de variables suele requerirse el conocimiento de XPath [59] así como también el conocimiento de XMLSchema [74].

Por otro lado, se puede observar que en la práctica la estrategia más común para composición de servicios es mediante el uso de lenguajes convencionales (Java o C#). La ventaja de esto es que los programadores no requieren conocer un nuevo lenguaje de programación ya que utilizan los mismos mecanismos que ya conocen del lenguaje (junto con algún marco de trabajo, como mencionamos en el apartado anterior). La desventaja es que estos lenguajes no fueron diseñados para la composición de servicios y las facilidades que ofrecen para ello suelen demandar del programador el conocimiento de muchos detalles tecnológicos, en lugar de tener un enfoque mayor en la solución del problema.

En cualquiera de los casos mencionados se distingue que el problema es el nivel de abstracción (cercano a la tecnología) que poseen los mecanismos disponibles para la composición de servicios que dificulta la labor del programador. En este sentido, la definición de primitivas conceptuales que permitan especificar esta composición en términos de más alto nivel sería muy valioso, permitiría centrarse en la definición de servicios que producen y consumen funcionalidad, así como en la composición de servicios complejos a partir de otros más simples y lograr así apoyar mejor la tarea del programador.

2.3 Conclusiones

Las aplicaciones Web de la siguiente generación deberán contar con diversas características: interfaces ricas, estéticas, que hagan uso de flujos de trabajos y con

capacidad de integración dada por los servicios Web [75,76]. La incorporación de las mismas a las aplicaciones Web requiere de una gran diversidad de tecnologías. Por ejemplo, las interfaces ricas y estéticas están siendo implementadas con Ajax : una combinación de XHTML, HTML, CSS, DOM, JavaScript y JScript. La situación es semejante para los flujos de trabajo y los servicios Web. Para el dominio de todas estas tecnologías se requiere de mucha experiencia así como de conocimiento de diversas tecnologías (su uso conjunto implica una curva de aprendizaje pronunciada), que suele no encontrarse fácilmente en la mayoría de los programadores. Aún dominándose, asegurar la calidad del software desarrollado no es tarea fácil por la complejidad que demanda su uso conjunto.

En este contexto, los métodos de Ingeniería Web pueden ofrecer soluciones para abordar la complejidad, con sus modelos, técnicas y herramientas orientadas al diseño conceptual. Sin embargo como hemos discutido en este capítulo, poco se ha hecho para la inclusión de estos aspectos en la mayoría de los métodos, salvo el caso de WebML. En algunos de los casos discutidos, las soluciones suelen brindar primitivas con un nivel de abstracción cercano a la tecnología.

Por otra parte, las tecnologías de servicios Web pueden también beneficiarse del diseño conceptual. El surgimiento de las Arquitecturas Orientadas a Servicios demandan una gran cantidad de tecnologías que deben utilizarse de manera conjunta y compleja. Construir este tipo de aplicaciones artesanalmente (usando las herramientas tradicionales) suele requerir de un gran esfuerzo. Asegurar su calidad es otro factor difícil de lograr. Si su diseño se realiza a nivel conceptual se puede facilitar la tarea de construcción y de aseguramiento de la calidad al diseñador. Otro beneficio es la protección que los modelos conceptuales brindan al futuro cambio tecnológico. En este contexto, es necesario ofrecer mecanismos de modelado conceptual que permitan su especificación y desarrollo.

Como se puede observar, ambas áreas se pueden ver beneficiadas con el modelado conceptual y convergencia. Lograr la convergencia, a este nivel, implica lograr la integración de sus modelos. Una mayor incorporación de modelos conceptuales de servicios Web en los métodos de Ingeniería Web ayudaría a las tareas de especificación, desarrollo y generación de las aplicaciones Web que se están demandando.

CAPÍTULO 3

Introducción al método OOWS

Una vez revisados algunos de los métodos más representativos de la Ingeniería Web el siguiente capítulo introduce el método OOWS, sobre el cual se enfoca el trabajo restante de esta tesis. Partiendo de su definición original (que permite la generación de aplicaciones Web centradas en datos), se menciona brevemente el conjunto necesario de adecuaciones y agregaciones a sus modelos y guías metodológicas, para que pueda, además, generar aplicaciones basadas en servicios Web. Ofrece el marco de trabajo general para los capítulos restantes.

El capítulo se organiza de la siguiente manera: en la primera parte se da una explicación general del método. Luego, se mencionan las características adicionales básicas que requiere para generar aplicaciones Web basadas en servicios Web. Posteriormente, se establece el conjunto de premisas que se considerarán al introducir estas nuevas características. Finalmente se ofrecen los pasos generales para lograrlo. Estos pasos generales se extienden al detalle en los capítulos posteriores.

3.1 OOWS

Esta primer sección presenta el método de Ingeniería Web *Object-Oriented Web-Solutions* (OOWS). Sobre éste se han definido las extensiones para el desarrollo de aplicaciones Web basadas en servicios Web, propuesta principal de esta tesis.

3.1.1 Panorama general

El método OOWS [9] es la extensión para el modelado conceptual de aplicaciones Web del método de producción de software OO-Method [10]. Los requisitos para este tipo de aplicaciones se capturan en un par de modelos -de *Navegación* y *Presentación*- que se agregan a los ya existentes en OO-Method: el *Modelo Estructural* (un Diagrama de clases), el *Modelo Dinámico* (un par de Diagramas de Estado y Secuencia) y el *Modelo Funcional* (una especificación textual tomado a partir de la especificación formal OASIS [10]) (ver Figura 3.1).

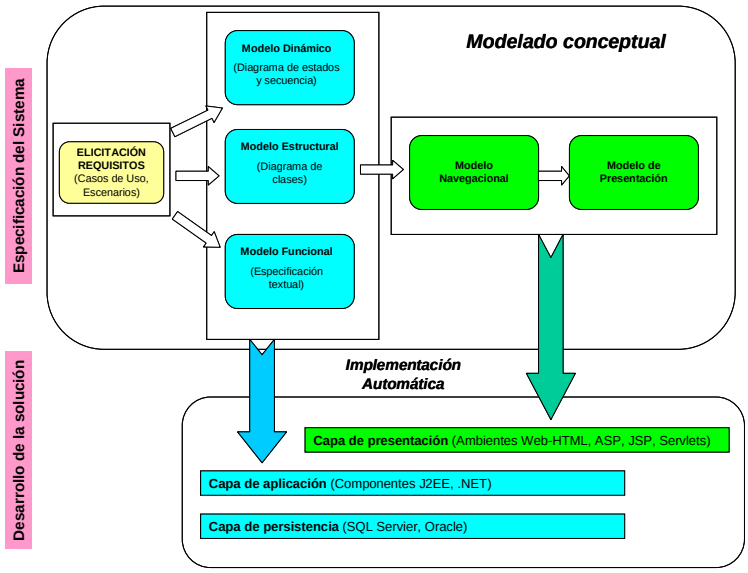


Figura 3.1 El método OOWS

El método está formado por dos fases principales: *Especificación del sistema* y *Desarrollo de la solución*. En la primera se construye el modelo conceptual que captura los requisitos estructurales y funcionales de la aplicación Web. El proceso de modelado se divide a su vez en dos pasos:

1. ***Elicitación de requisitos funcionales***: donde se identifican los requisitos funcionales mediante casos de uso y escenarios.
2. ***Modelado conceptual***: donde se capturan y representan los requisitos del sistema software desde los puntos de vista:

a. ***Estructural***: definiendo la estructura del sistema (sus clases, operaciones y atributos) y las relaciones entre clases (especialización, asociación y agregación) por medio de un *Diagrama de Clases*.
b. ***Dinámico***: describiendo los diferentes ciclos de vida de cada clase del sistema, usando *Diagramas de Transición de Estados*; así como también las interacciones entre objetos (sus comunicaciones), mediante *Diagramas de Secuencia*.
c. ***Funcional***: capturando la semántica de los cambios de estado de los objetos para definir el efecto de los servicios, usando una especificación textual formal [10].

En esta misma fase se capturan los requisitos de *navegación*, por medio de un *Diagrama de Usuarios* y un *Modelo Navegacional*; y los requisitos de *presentación*, por medio de un *Modelo de Presentación*.

En la segunda fase, se define la reificación del modelo conceptual OOWS a representaciones software por medio de dos pasos principales:

1. ***Definición del estilo arquitectónico***: en el cual se establece la arquitectura software de la aplicación, normalmente basada en un estilo de tres capas: presentación, aplicación y datos.
2. ***Definición de correspondencias***: entre las primitivas conceptuales que constituyen los modelos y componentes software que implementan cada capa de la arquitectura, haciendo uso intenso de patrones de traducción.

Se explica a continuación las primitivas conceptuales principales que capturan los requisitos de navegación y presentación.

3.1.2 Diagrama de Usuarios

Antes de iniciar el modelado de navegación se requiere construir un *Diagrama de Usuarios* para especificar los tipos de usuarios que podrán interactuar con el sistema, así como la visibilidad que tendrán sobre los atributos y operaciones de las clases. Este diagrama ofrece mecanismos para su gestión (tal como la especializa-

ción) que permiten definir taxonomías que posibilitan el reuso de la especificación navegacional [11].

Dependiendo de como se conectará el usuario al sistema existirán tres tipos (ver Figura 3.2):

- ***Anónimos:*** (representados con un símbolo '?' en su cabeza) para los cuales no es necesario autentificación.
- ***Genéricos:*** (representados con un símbolo 'X' en su cabeza) los cuales no podrán conectarse al sistema. Se representan como actores abstractos, por lo que requiere proveerse el usuario registrado concreto.
- ***Registrados:*** (representados con un símbolo de candado en su cabeza) los cuales necesitan autentificarse para conectarse al sistema. Representan un usuario concreto.

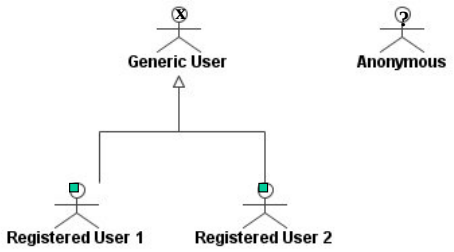


Figura 3.2 Diagrama de Usuarios

Cuando un usuario (*Anónimo* o *Registrado*) accede a la aplicación, tiene disponible una vista navegacional de las clases. Esta vista navegacional le permitirá su exploración consultando sus atributos e invocando sus operaciones. La vista navegacional se especifica mediante un *Mapa Navegacional* que se explica a continuación.

3.1.3 El Modelo Navegacional

El *Modelo Navegacional* posee la expresividad necesaria para capturar los requisitos de navegación de la aplicación Web. Este modelo se forma por una colección de *Mapas Navegacionales*, cada uno de los cuales representa una vista hipermedial que un usuario tiene sobre las clases, ofreciendo además semántica de navegación y presentación. El mapa es un grafo dirigido en el cual los nodos son llamados *Contextos Navegacionales* y las aristas *Enlaces Navegacionales.*

Cada *Contexto Navegacional* es una unidad de interacción que representa una vista cohesiva de datos y funcionalidad (atributos y operaciones de las clases), representado mediante un paquete UML que posee el estereotipo `<<Context>>`. Dependiendo de la forma en la que pueden ser alcanzados los contextos al navegar, se clasifican en uno de dos tipos: si se pueden alcanzar desde cualquier nodo se les llama *Contextos de Exploración* (indicado con la letra "E"), uno de los cuales se identifica como el Contexto de inicio (*Home* - indicado con la letra "H"). Si el contexto se alcanza a través de una secuencia de pasos de navegación (Enlaces Navegacionales), es llamado *Contexto de Secuencia* (indicado con la letra "S").

Los *Enlaces Navegacionales* representan pasos de navegación entre los contextos y se representan mediante asociaciones entre paquetes UML. La Figura 3.3 muestra un Mapa Navegacional para una Aplicación Web de comercio electrónico ilustrando usuario, contextos y enlaces.

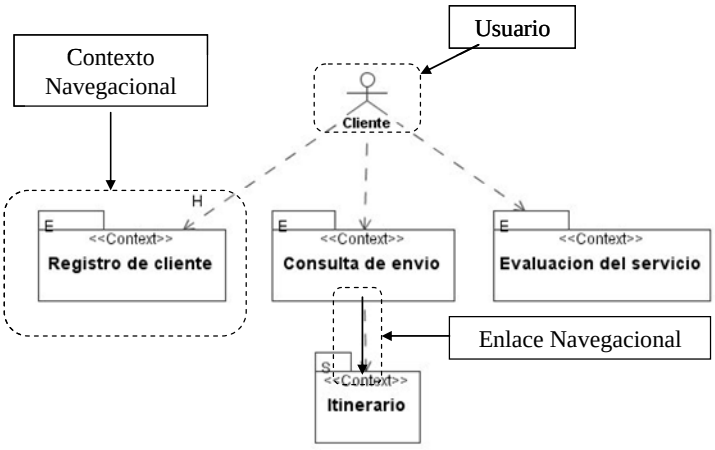


Figura 3.3 Mapa Navegacional para una tienda en-línea

La construcción del Mapa Navegacional se realiza en dos etapas: (1) *Autoreo-a-escala-mayor* (*Authoring-in-the-large*), en la cual se obtiene su diseño global definiendo los Contextos Navegacionales y los Enlaces de Navegación. La Figura 3.3 es un ejemplo del modelo que se obtiene en esta etapa. Y (2) *Autoreo-a-escala-menor* (*Authoring-in-the-small*), donde se obtiene el diseño detallado de cada uno de los Contextos Navegacionales.

Cada Contexto Navegacional se compone de *Clases Navegacionales*, representadas mediante clases UML que poseen el estereotipo `<<view>>`; así como de *Relaciones Navegacionales* que las enlazan. Cada Clase Navegacional se correspon-

den con alguna clase del Diagrama de clases OO-Method y constituye una vista de sus atributos y operaciones. De todas las Clases Navegacionales una es llamada *Directora* y constituye la información principal que se recupera para el Contexto Navegacional, el resto es un conjunto (posiblemente vacío) de Clases Navegacionales *Complementarias*, que complementan la información de la Clase Directora. En cada Clase Navegacional es posible especificar una *condición de filtro sobre la población* de los objetos de la clase. Esta condición define un conjunto de objetos que deben satisfacer la condición. Se describe mediante una fórmula OCL.

Las Relaciones Navegacionales se establecen para vincular las Clases Complementarias con la Clase Directora. Representan una relación binaria unidireccional definida sobre las relaciones estructurales (de agregación, asociación o especialización) del Modelo estructural. Las Relaciones Navegacionales pueden ser de dos tipos: de *Dependencia Contextual*, la cual posee semántica de recuperación de información complementaria; o de *Contexto*, que además agrega semántica de navegación a un contexto destino.

La Figura 3.4 muestra el resultado del diseño en la etapa de *Autoreo-a-escala menor* para los Contextos Navegacionales *Consulta de envío* e *Itinerario* de la Figura 3.3. El Contexto `Consulta de envío` tiene como Clase Directora a `Entrega`, con la cual se recuperan las entregas de artículos, ofreciendo además el servicio `obtEstadoActual()`, que permite consultar el estado actual de la entrega de un artículo. A su vez la Clase Complementaria `Itinerario` complementa la información con el itinerario de cada `Entrega`. Se enlazan con la Relación de Contexto (`Entrega-Itinerario`) que posee como atributo (también llamado *de contexto*) a `[Itinerario]`, que representa un Contexto Navegacional de Secuencia hacia el cual se navegará obteniendo el Itinerario correspondiente al visualizar el atributo `ruta`.

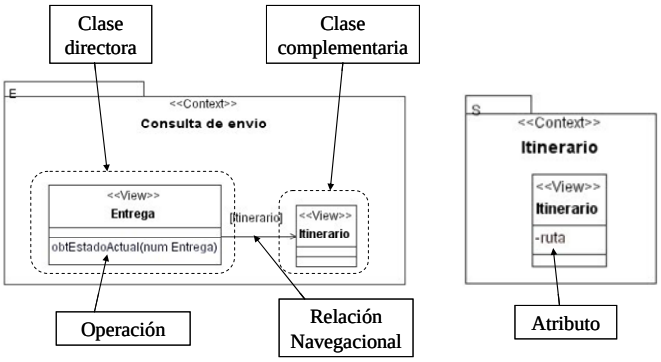


Figura 3.4 Contextos navegacionales y su diseño de Autores a escala menor

Otro requisito común de las aplicaciones Web (por ejemplo en portales) es la agregación de contenidos. Para su especificación, cada contexto navegacional puede estar formado por un conjunto de *Unidades Abstractas de Información (UAI)*. Una UAI representa un requisito particular de recuperación de información, independiente de otro que pudiera estar definido en otra UAI. Las UAI pueden ser de dos tipos: (1) *contextuales* (etiquetadas con un círculo C) las cuales se instancian cuando el sistema alcanza el contexto siguiente en un enlace secuencial portando información contextual y (2) *no contextuales* (etiquetadas con un círuclo NC) las cuales no dependen de enlaces secuenciales y no portan información contextual. Cada UAI está formada por clases navegacionales, tal como han sido expuestas anteriormente para los contextos navegacionales.

3.1.4 Características avanzadas de navegación

El número de instancias que se recuperan para un Contexto Navegacional, a partir del Diagrama de clases, es llamado *Cardinalidad*. La información recuperada se puede manejar mediante dos mecanismos que permiten su exploración y filtrado: *índices* y *filtros*.

Un *índice* es una estructura que ofrece acceso indexado a la población de objetos de la clase directora. Crea una lista sumarizada de información definida a partir de una o más propiedades (atributos). Si la propiedad indexada pertenece a la clase directora, se debe especificar como un atributo del índice. Si pertenece a una clase complementaria, el índice se define como un índice relacional y se debe especificar la relación. Al activarse el índice, se crea una lista con los valores de los atributos

indexados. Al seleccionar uno de estos valores, se muestran en una vista de búsqueda de todos los objetos que tienen el valor seleccionado para la propiedad. Esta vista de búsqueda muestra la información disponible para el usuario que le permitirá seleccionar una instancia. La instancia seleccionada será la activa en el Contexto Navegacional.

Un *filtro* define una condición sobre la población de instancias de objetos a ser recuperada. Se aplica a los atributos de la clase directora o a los atributos de las clases complementarias. Existen tres tipos: (1) *Exacto*, que toma un valor de atributo y regresa todas las instancias que empatan exactamente; (2) *Aproximado*, que toma un valor de atributo (una cadena) y regresa todas las instancias cuyos valores de atributos lo contienen como sub-cadena y (3) *Rango*, que toma dos valores (máximo y mínimo) y regresa todas las instancias cuyos valores de atributos están dentro del rango.

3.1.5 Modelo de Presentación

Los requisitos de presentación de los Contextos Navegacionales se capturan en el *Modelo de Presentación*. Estos requisitos se especifican por medio de un conjunto de patrones que se asocian a las primitivas del contexto:

- **De paginación de la información:** que permite definir la capacidad de desplazamiento vertical (*scrolling*) sobre la información. Todas las instancias se dividen en bloques lógicos, de tal forma que cada bloque es visible a la vez. Además se ofrecen mecanismos para avanzar y retroceder. El patrón se aplica a la clase directora, a la relación navegacional, a un índice o a un filtro. La información requerida por el patrón incluye: (1) la *Cardinalidad*, el número de instancias del bloque; (2) el *Modo de acceso*: *secuencial*, para acceder al bloque siguiente, previo, primero y último y *aleatorio*, para acceso directo o *circular*, para un comportamiento circular del conjunto de bloques.
- **De ordenamiento:** que define un criterio de ordenamiento de la población (con valores *ASC*=ascendente o *DESC*=descendente) acorde al valor de uno o más atributos. Se puede aplicar a clases navegacionales, a estructuras de acceso o mecanismos de búsqueda.

- **De distribución de la información:** con cuatro patrones básicos: *registro*, *tabular*, *maestro-detalle* y *árbol*. Se utiliza para indicar la forma en que se mostrará la información. Se puede aplicar a la clase directora o a las relaciones navegacionales.

3.2 Extensiones básicas para generar aplicaciones Web basadas en servicios

Tal como ya se ha discutido en el capítulo anterior, con relación a los métodos de Ingeniería Web, OOWS fue también diseñado originalmente para la especificación y generación de aplicaciones Web *centradas-en-datos*. Para extender sus capacidades, de tal manera que pueda además *integrar datos y funcionalidad* de aplicaciones externas, se requiere que las aplicaciones Web que especifica y genera, cuenten con la facultad de producir y consumir funcionalidad mediante servicios Web.

Bajo este requisito fundamental, la propuesta de esta tesis es que OOWS sea extendido para contar con las siguientes características:

- **Publicar parte de su funcionalidad:** correspondiendo a los aspectos funcionales de la aplicación. Los modelos conceptuales son la fuente principal para llevar a cabo esta tarea, junto con un método que defina la forma en la que se pueden publicar los aspectos capturados en los mismos. La publicación podría ser consumida por otras aplicaciones o formar parte de procesos Negocio-a-Negocio.
- **Importar funcionalidad:** definir modelos y métodos que permitan importar funcionalidad publicada por otras aplicaciones mediante servicios Web para ser incorporada como parte de la nueva aplicación Web.
- **Participar en colaboraciones distribuidas:** en cuyo caso la aplicación Web debería incluirse como un participante más de un proceso Negocio-a-Negocio, así como también como aplicación unificadora (orquestadora) del proceso.

3.3 Premisas de las extensiones

Para la implementación de las nuevas características mencionadas, se propone una estrategia general que considera la introducción de algunos modelos nuevos, así como adecuaciones a los ya existentes. Estos nuevos modelos y adecuaciones ofrecerán al diseñador la expresividad necesaria para la introducción de servicios Web en la especificación de la aplicación Web.

Las nuevas abstracciones conceptuales se han diseñado considerando las siguientes premisas:

- **Que se mantenga el principio de separación de aspectos:** cada abstracción deberá estar enfocada a la captura de aspectos claramente diferenciados, buscando evitar la mezcla de conceptos [50].
- **Que se mantenga la ortogonalidad:** los nuevos modelos deberían capturar requisitos nuevos no capturados en otros modelos existentes.
- **Buscar agregar un número mínimo suficiente de abstracciones:** el número de nuevas primitivas conceptuales debería ser el necesario suficiente, procurando evitar aumentar significativamente la terminología existente.
- **Lograr la integración con los modelos existentes:** las nuevas primitivas conceptuales deberían integrarse con los modelos ya existentes. Éstos podrían requerir alguna adecuación o extensión, la cual debería ser mínima y compatible con las nuevas primitivas.
- **Facilitar la generación automática de código:** las nuevas primitivas conceptuales deberían ser precisas, de tal manera que sea posible su correspondencia con representaciones software, sin ambigüedades.

La estrategia general incluye también la introducción de algunas guías metodológicas que permiten definir diversos aspectos en la construcción de los modelos. Así, el método establece guías para:

- **La identificación y construcción de operaciones públicas de los servicios Web de la aplicación:** realizado a partir del Diagrama de clases así como también a partir de los servicios Web (propios y

ajenos) de la aplicación. Las operaciones construidas pueden ser de dos tipos:

1. **De grano fino:** implementadas como vistas de las operaciones del Diagrama de clases cuya funcionalidad tiene correspondencia uno a uno con la funcionalidad de las operaciones (métodos) de las clases del Diagrama de clases.
2. **Operaciones de grano grueso:** operaciones compuestas a partir de las operaciones preexistentes en la aplicación (de grano fino o grano grueso) o a partir de operaciones de servicios Web ajenos a la aplicación, cuya funcionalidad se desea incorporar a la misma. Dicha composición puede ser especificada a nivel conceptual mediante relaciones de composición/agregación y especialización entre servicios.

- **La construcción de aplicaciones Web complejas a partir de procesos Negocio-a-Negocio:** partiendo de la definición de procesos Negocio-a-Negocio basados en servicios Web que muestran la interacción entre socios de negocio buscando obtener la aplicación Web que de soporte a la participación humana.

3.4 Pasos generales para la extensión y adecuación del método

Las adecuaciones y agregaciones de modelos y guías metodológicas, se incluyen como extensión en el método OOWS mediante los siguientes pasos:

1. **Introducción del modelado de servicios:** para la captura de los aspectos relativos a los servicios Web de la aplicación se define un *Modelo de Servicios*, adicional a los ya existentes en el método. Este modelo especifica los servicios propios (internos) y ajenos (externos) de la aplicación. A su vez permite definir aspectos estructurales entre servicios mediante relaciones que permiten capturar requisitos de composición (de agregación) así como mecanismos de reuso (de especialización). En algunos casos de composición, además, se requie-

re definir aspectos dinámicos , para lo cual se define un *Modelo Dinámico de Composición de Servicios*.

2. **Modificación del Modelo Navegacional:** se adecuan las primitivas del Modelo Navegacional con el fin de utilizar dentro del espacio hipermedial la nueva funcionalidad capturada en el Modelo de Servicios. Las primitivas no solo permitirán la captura de los requisitos de navegación sobre la colección de datos sino también posibilitarán la utilización de la funcionalidad especificada en el Modelo de Servicios, como parte del Modelo Navegacional.
3. **Extensión del Modelo de Presentación:** agregando patrones de presentación para la captura de estos requisitos derivados de la inclusión de los servicios Web en el Modelo Navegacional.
4. **Definición de Procesos Negocio-a-Negocio con participación humana**: en los cuales se considera la participación humana a través del Web en ciertos puntos del proceso. Partiendo de un proceso especificado como un Diagrama de Actividad UML y mediante guías metodológicas se genera un prototipo de la aplicación Web que de soporte al proceso.
5. **Transformación de modelos a representaciones software**: siguiendo el marco de trabajo establecida por la Arquitectura Dirigida-por-Modelos (MDA-*Model driven Architecture*) [36], se define cada modelo como un Modelo Independientes de Plataforma (PIM- *Platform Independent Model*), correspondido con uno o más Modelos Específicos de Plataforma (PSM-*Platform Specific Models*), a través de un conjunto de reglas de transformación definidas en lenguaje QVT operacional. Luego, estos Modelos Específicos de Plataforma son la base para la generación de código final a través de un conjunto de reglas de transformación de modelos a texto definidas en el lenguaje MofScript.

3.5 Conclusiones

Con las adecuaciones y extensiones propuestas se define un marco de trabajo que permitirá a OOWS poder generar aplicaciones Web basadas en servicios Web. Se han presentado las premisas en las que se basará este marco de trabajo así como los pasos necesarios para la extensión. En los siguientes capítulos se detallan estas adecuaciones y extensiones, así como también las correspondencias que transformarán los nuevos modelos a representaciones software.

CAPÍTULO 4

Modelado conceptual de servicios Web

El presente capítulo introduce las primitivas necesarias para el modelado conceptual de servicios Web, cuya sintaxis abstracta se especifica mediante un metamodelo. Las diversas secciones de este metamodelo se organizan en paquetes, a partir de los cuales se estructura también el presente capítulo de la tesis. El modelado conceptual de servicios Web corresponde a la extensión básica necesaria al método OOWS para lograr la especificación y captura de los aspectos de integración para la aplicación Web. Es el medio que posibilitará la especificación de los aspectos de producción y consumo de funcionalidad de la aplicación, con lo cual también se posibilita su participación en procesos Negocio-a-Negocio.

El capítulo está organizado en dos secciones principales: en la primera, se exponen los motivos para el modelado conceptual de servicios y su introducción en los métodos de Ingeniería Web; en la segunda, se presentan los elementos básicos del modelado de acuerdo a la organización en paquetes del metamodelo: primero, las metaclases básicas del paquete *Foundation*; luego, los elementos de modelado que capturan los aspectos estructurales y dinámicos del modelado de servicios, incluidos dentro de los paquetes *Structural* y *Dinamic*; finalmente, las metaclases del paquete *Management*, que permiten organizar los servicios Web dentro del modelo.

4.1 Motivación

El desarrollo de aplicaciones Web es una tarea compleja. Aspectos, como la navegación y la presentación, no presentes en los tipos de aplicaciones restantes, han llevado a concluir que el uso de los métodos clásicos de ingeniería de software para su construcción, no es suficiente y por tanto se requieren adecuaciones y nuevos métodos para su desarrollo [37]. Como respuesta a esta problemática, los métodos de la Ingeniería Web han definido modelos y técnicas principalmente orientadas para soportar el diseño de aplicaciones Web que consultan y actualizan grandes volúmenes de datos.

Aún cuando muchas de las aplicaciones Web utilizan datos locales, los modelos de negocio actuales demandan, además, integrar datos y funcionalidad de aplicaciones externas (ej. Tiendas electrónicas, sistemas de reservaciones de viajes, bibliotecas digitales corporativas, Sistemas de Planificación de Recursos –ERP- o Sistemas Gestores de la Relación Comercial con el Cliente –CRM-). La tendencia actual es que estas aplicaciones ofrezcan sus plataformas (datos y funcionalidad) a través de servicios Web y posibilitan con ello la integración con sus socios de negocio.

Aunque existen herramientas para el desarrollo de aplicaciones basadas en servicios Web (ej. GlassFish [38], WSTK [39]), el problema con las mismas es la gran cantidad de detalles tecnológicos que los desarrolladores deben considerar, lo cual dificulta su tarea.

Así, la alternativa que facilitar la tarea, estaría dado por primitivas independientes de la tecnología, que puede ofrecer el modelado conceptual.

En este contexto, se han definido diversos modelos conceptuales para la descripción de los diversos aspectos de los sistemas orientados a servicios: Colombo et al [77] ofrecen un modelo conceptual basado en UML que incluye roles, términos, relaciones y actividades (servicios, publicación, descubrimiento, composición, ejecución, monitoreo, actores, agentes). Gran parte de este modelo conceptual es semejante a la definición de servicios Web de la Arquitectura de Servicios Web (*Web Services Architecture* – WSA [78]), en la cual el modelado conceptual se divide en cuatro aspectos: servicios, mensajes, recursos y políticas. La desventaja de estos modelos conceptuales, para el problema planteado en esta tesis, es su complejidad. Muchos de los elementos de modelado que contienen podrían no ser requeridos para la especificación de las aplicaciones Web y de incluirse en un método de Ingeniería

Web podrían aumentar su complejidad de manera innecesaria. Bajo esta consideración, la propuesta que se ofrece en este apartado considera sólo un conjunto mínimo suficiente que permita especificar el Modelo de Servicios requerido.

Otras aproximaciones han seguido un enfoque basado en la descripción WSDL de los servicios Web ([79,80]). En este caso, se obtienen modelos UML mediante la importación de la descripción y posterior enriquecimiento con modelos estructurales y dinámicos, para la definición de nuevos servicios compuestos. El enfoque de estas aproximaciones considera fundamental la independencia tecnológica, con lo cual es posible la generación de implementaciones de servicios para diversas tecnologías. En contraste, algunos otras aproximaciones siguen un enfoque con alta dependencia de la tecnología. Por ejemplo, Provost [81] define perfiles UML para el modelado de servicios Web definidos a partir de las especificaciones WSDL y XML.

Un aspecto de interés importante adicional es la composición de servicios Web. Aquí se encuentra que la aproximación más común es su construcción mediante lenguajes de programación comúnes (como Java o C#). Las bibliotecas de sus plataformas se extienden para incluir las rutinas *stub* necesarias para ofrecer la lógica de programación necesaria adicional para la composición. Existen también lenguajes de programación definidos exclusivamente para la composición de servicios Web (como BPEL o BPML). En cualquier caso, de nuevo, el programador tiene que enfocarse en múltiples detalles técnicos que suelen dificultar su labor.

En constraste, la *Arquitectura de Componentes de Servicios* (ACS)[7] [105], es una opción que busca la independencia tecnológica. ACS provee un modelo de programación para la creación de componentes de servicios y un modelo para su ensamblado. Posee también un medio para el manejo de datos, independiente a su fuente o manejo (*Objetos de Datos de Servicio*[8]). Con ambos, se implementa el modelo de programación de la *Arquitectura Orientada a Servicios*. Si bien las primitivas que ofrece ACS para la composición ofrecen cierto grado de independencia tecnológica, ésta no es completa, puesto que siguen demandando del programador conocimiento de Java o XML.

Aunando lo presentado en la revisión del estado del arte, se tiene entonces un escenario en el cual se encuentran: (1) métodos enfocados principalmente al desa-

[7] En inglés *Service Component Architecture* (SCA).
[8] En inglés *Service Data Object* (SDO)

rrollo de aplicaciones Web basadas en datos locales, que no toman en cuenta los requisitos de integración demandados para las aplicaciones actuales de negocio electrónico; (2) herramientas para la construcción y composición de servicios Web que no proveen mecanismos a nivel conceptual, que incrementen la productividad y facilitan con ello el desarrollo de aplicaciones Web basadas en servicios Web y (3) soluciones de modelado conceptual de servicios Web que intentan modelar todos los aspectos de los sistemas orientados a servicios y que de incluirse en un método de Ingeniería Web aumentarian innecesariamente su complejidad.

Continuando entonces con el método OOWS, en las siguientes secciones se introduce la extensión que le posibilita el modelado conceptual de servicios Web, la cual retoma algunos de los mecanismos de modelado conceptual mencionados y agrega otros más, seleccionando sólo aquellos que pueden brindar solución al problema planteado.

4.2 El metamodelo de servicios

Los aspectos de producción y consumo de funcionalidad se capturan, en el método OOWS, en un *Modelo de Servicios*. Este modelo es traducido después a modelos específicos de plataforma, a partir de los cuales se genera el código final.

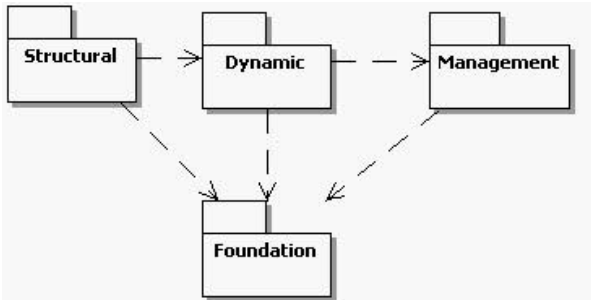


Figura 4.1 Paquetes principales del metamodelo de servicios

La Figura 4.1 muestra los paquetes principales (y sus dependencias) que conforman el metamodelo del Modelo de Servicios: (1) el paquete *Foundation*, que incluye las metaclases básicas del metamodelo, así como los tipos de datos; (2) el paquete *Structural*, que incluye las metaclases necesarias para establecer relaciones de agregación/composición y especialización entre servicios; (3) el paquete *Dynamic*, que incluye las metaclases necesarias para la especificación del comportamien-

to de operaciones de servicios compuestos; y (4) el paquete *Management*, que posee las primitivas arquitectónicas necesarias para la organización de los servicios en grupos funcionales, que permiten la organización del modelo. A continuación se describe cada paquete.

4.2.1 Aspectos básicos : el paquete *Foundation*

El paquete *Foundation* posee las metaclases básicas del metamodelo. Se organiza a su vez en dos paquetes (ver Figura 4.2):

1. **Kernel:** que posee metaclases correspondientes a la funcionalidad que produce y consume la aplicación. Esta funcionalidad está representada por metaclases de servicios con operaciones y parámetros.
2. **DataType:** que posee los tipos de datos que se utilizarán por los diversos elementos de modelado (ej. los parámetros de las operaciones o sus valores de retorno).

A continuación se explica al detalle ambos paquetes.

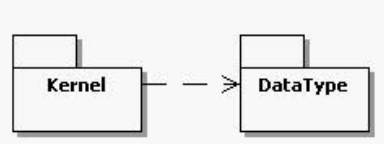


Figura 4.2 El paquete *Foundation*

4.2.1.1 El paquete *Kernel*

La funcionalidad que produce la aplicación está dada por un conjunto de Servicios propios (*Own-service*) (ver Figura 4.3). La funcionalidad que consume la aplicación está dada por un conjunto de Servicios ajenos (*External-service*). Esta división de la funcionalidad es necesaria ya que cada caso tiene consideraciones y tratamientos diferentes en aspectos como su definición, utilización, composición y especialización (que se explicarán más adelante). La razón principal está fundamentada en el hecho de que el diseñador tiene control y posibilidad de definición sólo en la funcionalidad propia y no en la funcionalidad ajena.

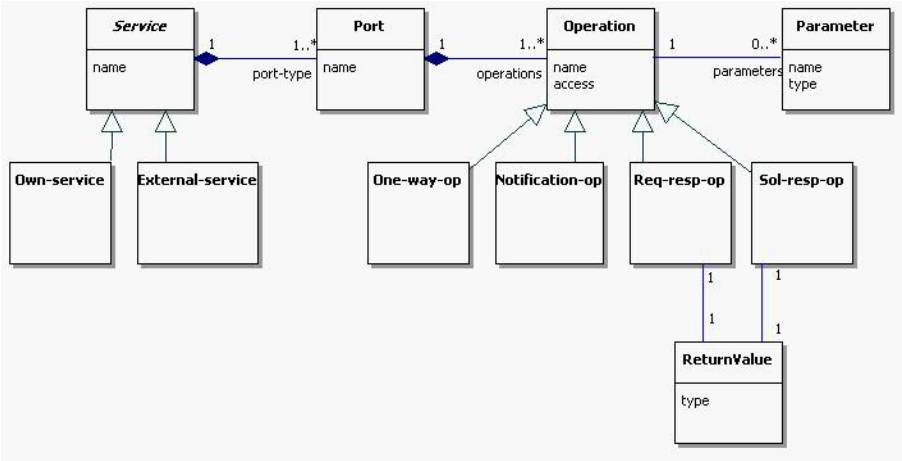


Figura 4.3 El paquete *Kernel*

Los Servicios propios se pueden construir como una vista de alguna de las operaciones existentes de las instancias de clases (del Diagrama de clases) y también por composición, usando las operaciones publicas de otros servicios (propios o ajenos). Los Servicios ajenos se obtienen mediante un proceso de importación *texto-a-modelo* a partir de la descripción textual de los mismos. Por ejemplo, en el caso de que el servicio ajeno haya sido implementado como un servicio Web, se requerirá una importación *WSDL-a-modelo*.

Cada servicio posee uno o más puertos (*Ports*). Cada uno de los cuales es un punto de acceso que ofrece un conjunto de operaciones para un uso particular. Estos puertos a su vez contienen una o más de los siguientes tipos de operaciones (*Operation*):

1. **One-way** (*One-way-op*): una operación asíncrona del servicio invocada por el cliente, la cual recibe cero o más parámetros de entrada y no devuelve respuesta.
2. **Notification** (*Notification-op*): una operación asíncrona de un cliente invocada por el servicio, la cual recibe cero o más parámetros de entrada y no devuelve respuesta.
3. **Request-response** (*Req-resp-op*): una operación síncrona del servicio invocada por el cliente, la cual recibe cero o más parámetros de entrada y devuelve una respuesta.

4. **Solicit-response** (*Sol-resp-op*): una operación síncrona del cliente invocada por el servicio, la cual recibe cero o más parámetros de entrada y devuelve una respuesta.

Cada parámetro (*Parameter*) y valor de retorno (*ReturnValue*) de cada operación puede ser de algún tipo de dato definido dentro del paquete *DataType*, que se explica a continuación.

4.2.1.2 El paquete *DataType*

Este paquete (Figura 4.4) posee los tipos de datos utilizados por algunos de los elementos de modelado (como los parámetros y los valores de retorno de las operaciones). Los tipos de datos son compatibles con los tipos de datos MOF y se clasifican en dos tipos principales:

1. **Simple**: que incluye a los tipos de datos primitivos, así como también las clases de los objetos de negocio[9] para el caso de Servicios propios; así como también por los tipos de datos importados a partir de los servicios ajenos. Estos tipos de datos se importan como tipo `Class` cuando no son compatibles con los tipos de datos simples existentes. Los tipos de datos primitivos son los siguientes:
    a. **Boolean**: para representar valores de verdad.
    b. **Integer**: para representar valores enteros con signo.
    c. **Float**: para representar valores de punto flotante.
    d. **String**: para representar cadenas de caracteres.

2. **Constructor**: los cuales permiten la definición de colecciones de tipos de datos. Pueden ser a su vez:
    a. **Structured**: un tipo de dato tupla que consiste de uno o más campos de un cierto tipo de dato.
    b. **Collection**: un tipo de dato cuyo valor es una colección finita de datos del mismo tipo.

[9] Es decir, cada uno de las clases del Diagrama de Clases es en si un tipo de dato que puede ser utilizado como parte del Modelo de Servicios.

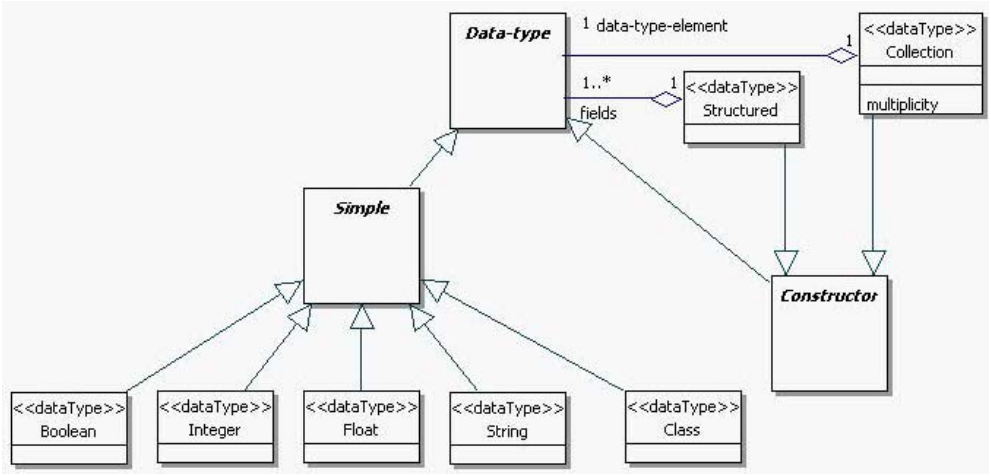


Figura 4.4 El paquete *DataType*

4.2.2 Aspectos estructurales : el paquete *Structural*

Otro de los aspectos que se consideran en el Modelo de Servicios es la composición de servicios. Mediante ésta se obtiene un nuevo Servicio propio (llamado *compuesto*) a partir de otros Servicios propios o ajenos (llamados *componente*). La composición depende de los tipos de servicios componente (propios o ajenos), cuyas operaciones se utilizan para definir la composición.

Se consideran dos aspectos complementarios para la especificación completa de la composición: su *estructura* y su *dinámica*. Los aspectos estructurales, porque con ellos se capturan las propiedades relativas al enlace y comunicación entre servicios, así como el reuso de funcionalidad mediante especialización. Los aspectos dinámicos, porque en ellos se define la lógica de ejecución de cada una de las operaciones del servicio compuesto. Para los aspectos estructurales se definen relaciones de agregación y especialización entre servicios, así como propiedades en las relaciones que permiten su definición precisa y facilitan la generación automática de código.

Las relaciones de agregación y especialización se establecen de la siguiente manera:

1. **Agregación:** al definir un Servicio propio (*compuesto*) a partir de la composición con otros Servicios propios o ajenos (*componentes*).

2. **Especialización:** al definir una relación de especialización (herencia simple) entre un Servicio propio o ajeno (*padre* o *base*) y las operaciones de un Servicio Propio (*hijo* o *derivado*). La especialización define nuevas operaciones en un Servicio propio hijo a partir de un Servicio propio o ajeno padre. El Servicio propio *hijo* define sus nuevas operaciones mediante (1) la redefinición o extensión de la lógica de alguna operación del Servicio propio o ajeno *padre*; o (2) la agregación de una o más operaciones en el Servicio propio o ajeno *padre*.

Para los aspectos dinámicos, se introduce además un modelo que permite definir la lógica de ejecución cada una de las operaciones del servicio compuesto.

A continuación se detallan ambos aspectos.

4.2.2.1 Agregación de servicios

La relación estructural en la composición de un Servicio propio compuesto con otros servicios componente, se define mediante relaciones de agregación. Esta relación es definidia extendiendo el metamodelo de servicios para incluir algunas abstracciones adicionales (ver Figura 4.5).

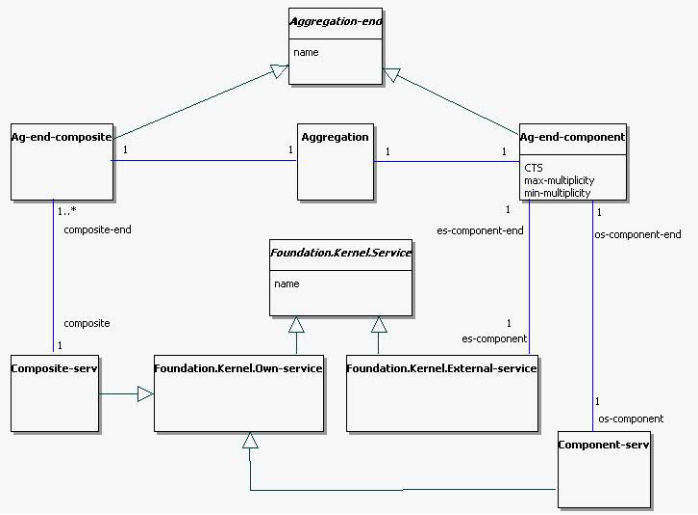


Figura 4.5 Metamodelo para agregación de Servicios

En este metamodelo, el Servicio propio agregando es llamado *Servicio Compuesto* (*Composite Service*) y los servicios propios o ajenos que se agregan son llamados *Servicios Componente* (*Component Service*).

Cada relación de agregación (*Aggregation*) posee en su extremo final (*Agend-component*) el conjunto de propiedades que la caracterizará. Este conjunto es explicado más adelante, así como también la forma en la que el metamodelo se enriquece y define con mayor precisión mediante un conjunto de restricciones OCL.

4.2.2.2 Especialización de servicios

La *especialización*[10] es otro mecanismo que puede utilizarse para la definición de nuevos Servicios propios. La especialización define un nuevo servicio propio a partir de otros Servicios propios o ajenos ya existentes.

En la literatura[43] encontramos cuatro tipos de relación entre los atributos y operaciones (llamados de forma general *propiedades*) de una clase base (o general) hacia una clase derivada (o especializada). Estas relaciones se han adoptado para la especialización de servicios.

Dado que en el Modelo de Servicios propuesto se tienen como propiedades solamente *operaciones*, en ese contexto se discute a continuación la caracterización de la especialización.

Propiedad heredada

Las operaciones del Servicio propio o ajeno que se especializará son heredadas en el nuevo servicio propio. En este caso la relación de especialización permite que las operaciones de los Servicios propios o ajenos se puedan incluir como parte de las operaciones que ofrecerá el Servicio Propio especializado.

[10] La *especialización* y la *generalización* son dos puntos de vista diferentes de la relación *Es-Un* (*Is-a*) entre dos clases (superclase y subclase). Aunque existe la especialización a partir de más de una superclase, la presente tesis se limita especialización simple (una superclase).

Propiedad modificada

La operación del Servicio propio o ajeno es *refinada* en un nuevo Servicio propio. Este refinamiento toma en cuenta el origen de la operación a especializar: si proviene de un Servicio ajeno es entonces una *caja negra* cuyo refinamiento estará más limitado (dado que no se tiene control sobre la misma); que si proviene de un *Servicio propio* (dado que sobre ella si se tiene control). En este sentido se consideran dos casos posibles:

1. **Refinamiento de la firma (signatura):** consiste en un refinamiento sintáctico de la firma de la operación, poseyendo uno o más parámetros adicionales. Se aplica para especializar operaciones de Servicios propios o ajenos. Siguiendo el principio *Abierto-Cerrado* (*Open-closed*) de Meyer [44], el refinamiento se realiza solamente *añadiendo comportamiento*. Así, la implementación utiliza la lógica de la operación padre, añadiendo lógica propia.

   Por ejemplo, supóngase que se tiene un Servicio ajeno (`TravelService`) para gestionar servicios de viajero el cual ofrece una operación (`getTripDetails`), que posee la funcionalidad de consulta/reservación de rutas de vuelos. El Modelo de Servicios estaría dado por la Figura 4.6.

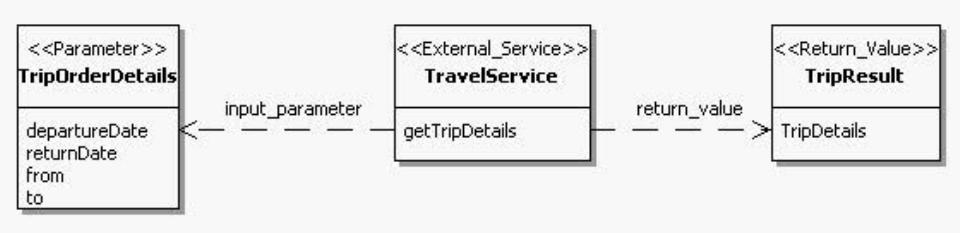


Figura 4.6 Modelo de servicios para consulta y reservación de viajes

A partir de esta funcionalidad se ha decidido construir un Servicio propio que ofrezca, además de la consulta/reservación de vuelos, la posibilidad de consultar/reservar hoteles. Esto implica que la operación disponible para el Servicio ajeno (`getTripDetails`) requiera adicionalmente un parámetro con los requisitos de hotel del usuario (`hotelOrderDetails`). Mediante el refinamiento de la firma (y de la lógica de operación, que se verá más ade-

lante) se pueda satisfacer el requisito planteado, tal como lo muestra la Figura 4.7.

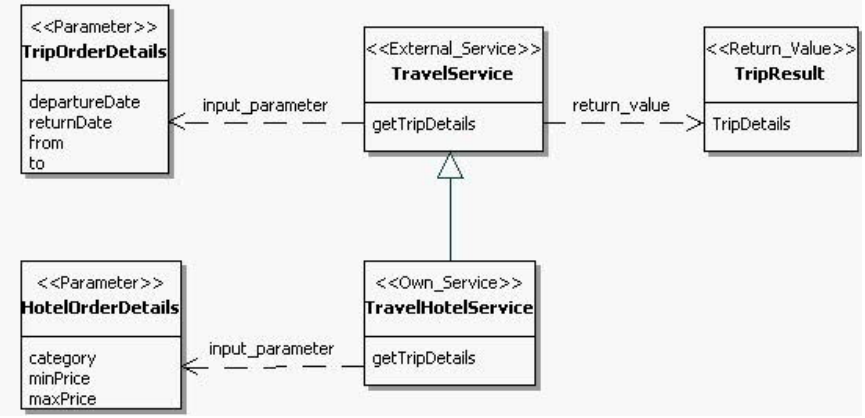


Figura 4.7 Especialización de servicios mediante refinamiento de la firma de una operación

2. **Refinamiento de la lógica de composición de la operación:** este caso aplica para operaciones de Servicios propios cuya composición ha sido previamente especificada mediante modelado dinámico[11]. En este caso la sintaxis de la firma de la operación del Servicio propio (base) no es modificada pero la lógica de su composición se refina. Este refinamiento se aplica para especializar operaciones de Servicios propios y ajenos teniendo un carácter diferente en cada caso.

   Para los Servicios propios la modificación consiste en refinar la lógica de la operación mediante dos mecanismos: (1) la identificación de pasos *virtuales* en la operación padre (etiquetados con la palabra clave `<<virtual>>` o como un valor etiquetado (*tagged value*) en la definición de un *perfil*) los cuales pueden ser redefinidos en la operación especializada o (2) mediante la inclusión de pasos adicionales en la lógica de la operación padre.

   Por ejemplo, la lógica de la operación para consulta/reservación de viajes y hoteles (`getTripDetails`) en el Servicio propio especializado (`Tra-`

[11] Esta lógica se captura en el *Modelo Dinámico de Composición de Servicios* (*MDCS*) que se explica más adelante.

velHotelService – ver Figura 4.7) se ha definido de tal forma que la tarea de reservación se corresponde con la invocación de una operación de reservación de hoteles (`getHotelDetails`) del servicio `Hotelkey`. Esta operación ha sido etiquetada como `<<virtual>>`. Esto posibilita su especialización en un *Servicio propio* especializado en el cual la reservación de hotel se podría solicitar a otro servicio. Por ejemplo, la Figura 4.8 redefine la operación mencionada con la operación `getHotelDetails` del Servicio externo `BusinessTraveler`.

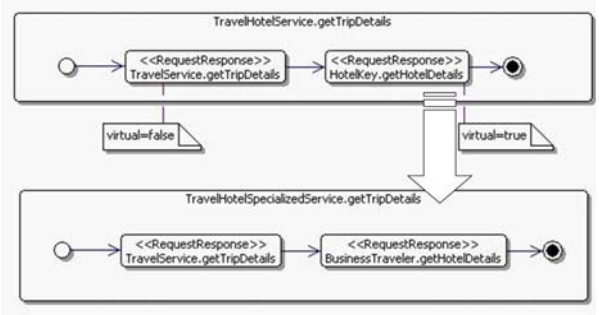


Figura 4.8 Especialización de pasos en la lógica de una operación

Para la especialización de operaciones de Servicios ajenos no es posible la redefinición de alguno de sus pasos, al no tenerse control sobre la lógica de la operación. En este caso la alternativa posible para la operación especializada consiste en utilizar la implementación ya existente y *añadirle nuevos pasos*. Esta es la estrategia que se sigue en la especialización de la operación `getTripDetails` del Servicio `TravelHotelService` en la Figura 4.9.

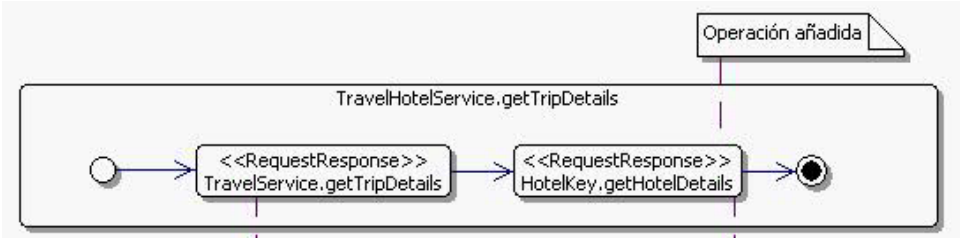


Figura 4.9 Especialización de la lógica de una operación añadiendo operaciones

3. **Propiedad nueva:** se obtiene un nuevo Servicio propio especializado añadiendo una operación al conjunto de operaciones del *Servicio propio* o *ajeno* que se está especializando.

Por ejemplo, suponga que se desea especializar el Servicio ajeno `TravelService` (ver Figura 4.6) para un nuevo servicio que, además de posibilitar la consulta/reservación de vuelos, permita también realizar reservaciones en restaurantes de cierta categoría y tipo de comida. En el Servicio propio (`TravelRestaurantService`) solamente se tendría que incluir la operación de reservación de restaurante (`reserveRestaurant`) y heredar del Servicio ajeno mencionado (ver Figura 4.10).

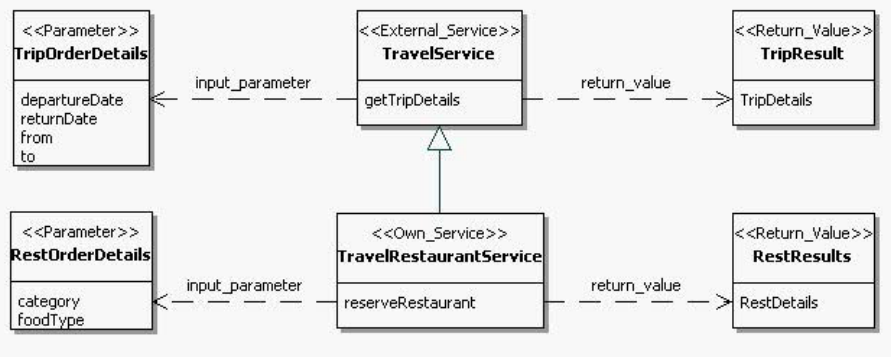


Figura 4.10 Especialización de Servicio externo agregando una operación

4. **Propiedad cancelada:** en la cual el Servicio propio podría borrar (olvidar) alguna de las operaciones del Servicio propio o ajeno a especializar. No se considera útil este caso.

Los aspectos estructurales, de los nuevos servicios, han sido definidos a partir de las relaciones mencionadas (agregación y especialización). Para que la especificación de los modelos sea completa y precisa se agregan algunas restricciones semánticas que se detallan a continuación.

4.2.2.3 Modelos precisos : definición de restricciones

En las relaciones estructurales de la agregación de servicios se especifican los requisitos referentes a la vinculación (*binding*) entre el Servicio propio compuesto y sus servicios componente. El metamodelo introducido en la sección 4.2.2.1 requerirá una definición precisa que facilite la tarea de generación automática de código así como de validación de modelos, por lo que se hace necesario la definición de algunas restricciones adicionales.

Siguiendo la propuesta del marco de trabajo (*framework*) multidimensional propuesto en [45], para la caracterización y definición precisa de propiedades de relaciones de agregación, se han identificado y seleccionado aquellas que representan características relevantes para la composición de servicios. Las restricciones sobre las propiedades se expresan en lenguaje OCL[46] (con respecto al metamodelo MOF en la Figura 4.5):

1. **Comportamiento temporal:**
    a. *Definición:* indica si el servicio compuesto tiene vinculación permanente o no a lo largo de toda su vida con el servicio componente.
    b. *Definido sobre:* extremo final de la agregación componente (*Ag-end-component*).
    c. *Nomenclatura:* $CTS_{Ag\text{-}end\text{-}component}$
    d. *Valores:* `Static`| `Dynamic`
        i. *Static:* el servicio componente está vinculado con el servicio compuesto durante toda su vida.
        ii. *Dynamic:* el servicio componente se selecciona dinámicamente a partir de valores de datos que se obtienen durante le ejecución de la lógica de composición (llamadas *variables de proceso*[15]). Comúnmente a partir de un directorio (UDDI[31] por ejemplo).
    e. *Restricción:*

```
context Ag-end-component
inv comportamiento-temporal:
CTS='Static' or CTS='Dynamic'
```

2. **Multiplicidad:**
    a. *Definición:* indica el número mínimo y máximo de servicios componente (que ofrecen la misma funcionalidad) vinculados con el servicio compuesto.
    b. *Definido sobre:* extremo final de la agregación componente (*Ag-end-component*).
    c. *Nomenclatura:* $Min_{Ag\text{-}end\text{-}component}$, $Max_{Ag\_end\text{-}component}$

d. *Valores:* enteros positivos.

e. *Restricción:*

```
context Ag-end-component
inv Multliplicidad:
 if os-component->size()> 0 then
      not (Aggregation.Ag-end-
      composite.composite->
      includes(os-component))
 else  true
 endif
```

Estas restricciones poseen algunas implicaciones adicionales que se mencionan a continuación:

1. Toda agregación incluye como Servicio componente a un Servicio propio (distinto al Servicio compuesto) o a un Servicio ajeno. Esta restricción se incluye en la siguiente invariante OCL:

```
context Ag-end-component
inv Al-menos-un-servicio-como-
componente:
os-component->size()>0 xor
es-component->size()>0
```

2. El valor `Static` para la propiedad de *Comportamiento temporal* implica que los valores máximo y mínimo para la propiedad *Multiplicidad* en la relación de agregación sean 1. Esto conocimiento se incluye en el modelo mediante la siguiente invariante:

```
context Ag-end-component
inv multiplicidad-estatica:
  CTS='Static' implies
  max-multiplicity=1 and min-
multiplicity=1
```

3. El valor `Dynamic` para la propiedad de *Comportamiento temporal* implica que el valor de multiplicidad máxima para la relación de agregación sea mayor que 1. El Servicio compuesto podría estar vinculado con dos o más posibles Servicios componente, donde la vinculación con uno solo (en un cierto momento en el tiempo) estará dado por *selección dinámica*, tal como será explicado más adelante. La restricción está dada por:

```
context Ag-end-component
inv dinamyc-multiplicity:
CTS='Dynamic' implies
Max-multiplicity > 1.
```

Este tipo de restricciones –y sus implicaciones- pueden ser útiles para la construcción de herramientas de modelado que incluyan verificación para ayudar al diseñador en la construcción correcta de sus modelos de servicios.

4.2.2.4 Ejemplo de agregación estática de servicios

Con el fin de ilustrar los conceptos de agregación y sus restricciones, se exponen a continuación algunos ejemplos. La Figura 4.11 muestra un ejemplo de Servicio propio componente (`BestStoreService`) para una aplicación de *e-Negocio* de venta de libros. Esta aplicación se ha sido definida a partir de la agregación de los Servicios Web de los sitios Amazon [47] y Barnes&Noble [48]. El nuevo Servicio propio poseerá operaciones para consultar la mejor tienda en base al mejor precio.

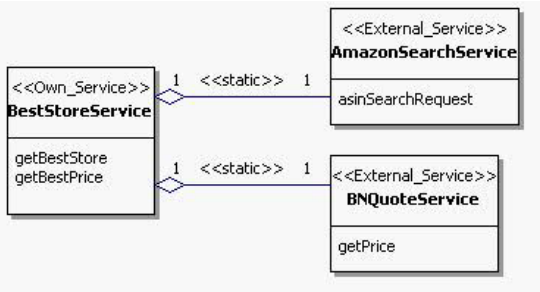


Figure 4.11 Ejemplo de Modelo de Servicios usando agregación estática

Dado que las operaciones para `BestStoreService` solamente utilizarán este par de servicios Web, la relación de agregación entre servicios se define con una propiedad de *Comportamiento temporal* con valor *Static*. Nótese que de acuerdo a la restricción 2, la multiplicidad máxima y mínima debe ser igual a 1.

4.2.2.5 Ejemplo de agregación dinámica de servicios

La Figura 4.12 muestra un ejemplo de agregación dinámica de servicios. En este caso, el Servicio propio compuesto (`SService`) es para una aplicación de proveedores. Este servicio se utiliza como parte del siguiente proceso: para la solicitud del precio de un producto, el servicio compuesto ofrece la operación (`getPriceInStock`). Esta operación primero verifica la existencia en el almacén local utilizando un Servicio Propio (`LWService`). Si el producto no está en existencia en el almacén local, entonces se verifica si existe en alguno de dos almacenes centrales, cada uno de los cuales posee, respectivamente, los Servicios ajenos `EWService1` y `EWService2`. Ambos son especializaciones del servicio abstracto `EWService` (ver Figura 4.13) y ofrecen funcionalidad semejante. Tal como se expone más adelante, esto es necesario para la selección dinámica de servicios.

La relación de agregación entre el Servicio compuesto (`SService`) y el Servicio propio (`LWService`) es definida *Static*. También la relación de agregación entre el Servicio compuesto (`SService`) y el Servicio ajeno abstracto (`EWService`) es definida *Dynamic*. Para que los servicios concretos `EWService1` y `EWService2` puedan ser utilizados dinámicamente, es necesaria su importación al Modelo de Servicios con algunas consideraciones que se explican a continuación.

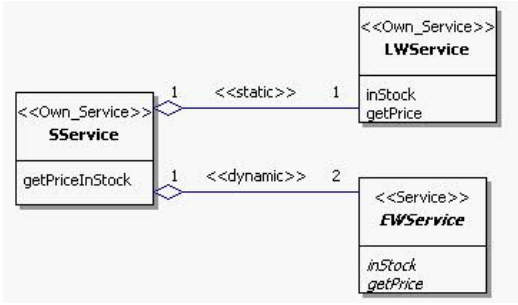


Figura 4.12 Ejemplo de Modelo de Servicios usando agregación dinámica

Cuando un conjunto de servicios ajenos será manejado dinámicamente deberán importarse al Modelo de Servicios a partir de información de los mismos proporcionada en algún registro UDDI (ver Figura 4.13). Este proceso de importación deberá incluir, además de los Servicios, una clase fachada (`FEWService` para este ejemplo) la cual posee una operación para la selección del servicio (`getXService`). La responsabilidad de esta operación es la selección dinámica del servicio concreto en base a una condición que se define en la operación de consulta OCL.

Esta condición se puede establecer en la definición del Modelo de Servicios o a partir de una expresión definida en el Modelo Dinámico de Composición de Servicios (explicado más adelante). En el primer caso, la condición se incluye en el Modelo de Servicios y se basa en los diferentes elementos de modelado de la aplicación. Por ejemplo, si un nuevo servicio Web (`EWService3`) -para un nuevo almacén central- se agrega al Modelo de Servicios, entonces la selección dinámica del servicio continua en este modelo y la definición del Modelo Dinámico de Composición de Servicios (explicando posteriormente) no cambiará. En el segundo caso, el modelador define una condición basada en las variables del Modelo Dinámico de Composición de Servicios la cual se pasará como parámetro de la operación `getXService` en el Modelo de Servicios. Más adelante, cuando se explique al detalle el Modelo Dinámico de Composición de Servicios, se ilustrará este mecanismo.

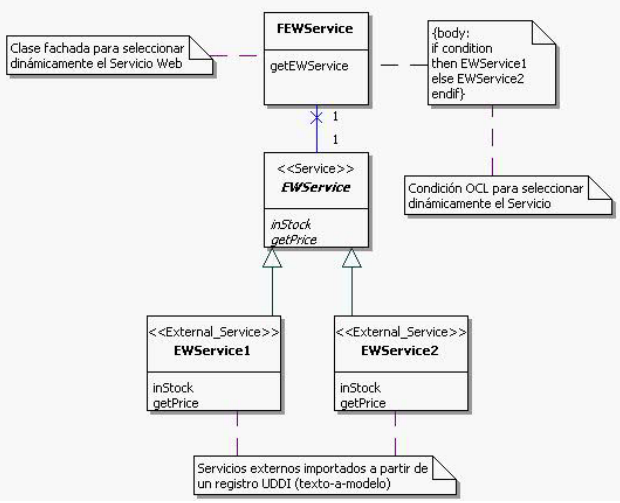


Figura 4.13 Importación de Servicios Web al Modelo de Servicios para uso dinámico

4.2.3 Aspectos dinámicos : el paquete *Dynamic*

La lógica de ejecución de cada una de las operaciones del Servicio compuesto se captura en un *Modelo Dinámico de Composición de Servicio (MDCS)*. Este modelo se define como un Diagrama de Actividad UML 2.1 cuyas acciones definen los diferentes pasos para el control e invocación de las operaciones de los servicios componente. El MDCS puede transformarse a alguna implementación tecnológica del servicio compuesto (como WS-BPEL, tal como se expone más adelante).

4.2.3.1 El metamodelo del MDSC

La Figura 4.14 muestra el metamodelo del MDCS, definido dentro del paquete *Dynamic*.

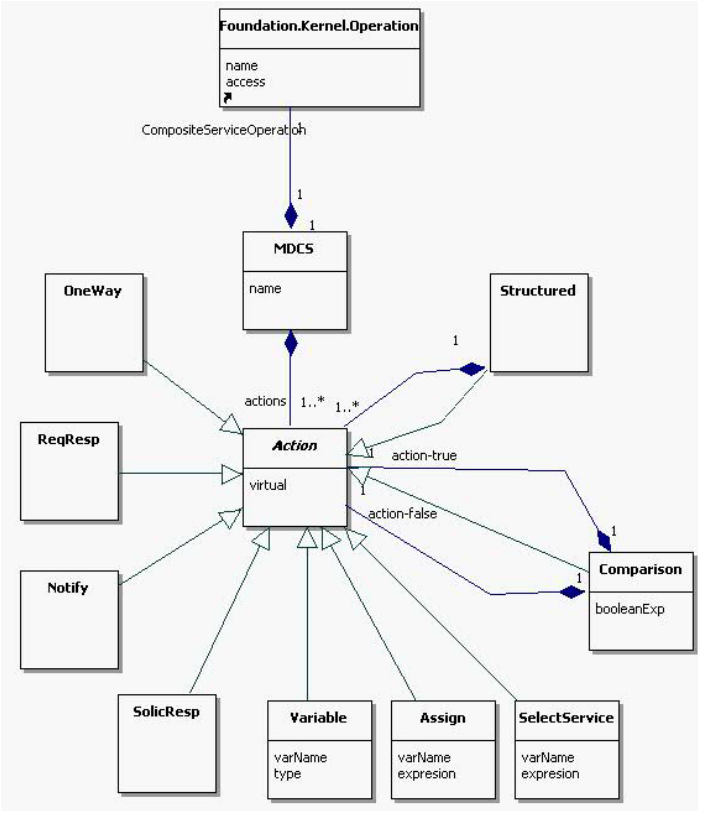


Figura 4.14 El metamodelo MDCS

Se define un MDCS para cada operación de un nuevo Servicio propio que se obtiene a partir de las relaciones estructurales antes expuestas. Los parámetros de entrada y el valor de retorno del MDCS corresponden a los valores de entrada y salida de la operación. Posee, además, un conjunto de acciones cada una de las cuales posee la propiedad *virtual* (de tipo *booleano* y con valor por defecto *falso*) para implementar el *Refinamiento de la lógica de composición de la operación* (apartado 4.2.2.2).

El modelo se define de forma precisa mediante restricciones OCL del siguiente tipo:

1. *La variable utilizada (varName) en la acción de asignación (Assign) debe estar previamente declarada en una acción (Variable)*:

```
context Assign
inv:
MDCS.Variable.varName-
>exists(x|x=self.varName)
```

2. *La variable utilizada (varName) en la acción de selección de servicio (SelectService) debe estar previamente declarada en una acción (Variable)*:

```
context SelectService
inv:
MDCS.Variable->
exists(x|x.varName=self.varName and
x.type='Service')
```

3. *En la acción de control de comparación (Comparison), la acción a realizar ante la condición booleana debe ser única. Si se desea realizar más de una acción debe utilizarse una acción estructurada(Structured). La ejecución de una acción ante la evaluación a un valor de (false) en la condición booleana es opcional:*

```
context Comparison
inv:
```

```
action-true->size()=1 and
action-false->size() <= 1
```

4. *No deben declararse variables dentro de una acción de comparación (Comparison):*

```
context Comparison
inv:
(not action-true->include(Variable) and
not action->false->include(Variable))
```

4.2.3.2 Ejemplo de definición de operación mediante el MDCS

La operación `getBestStore` para el Servicio Propio `BestStoreService` (ver Figura 4.11) la cual selecciona la tienda que ofrece el precio menor para un cierto libro, define su lógica mediante el MDCS de la Figura 4.15.

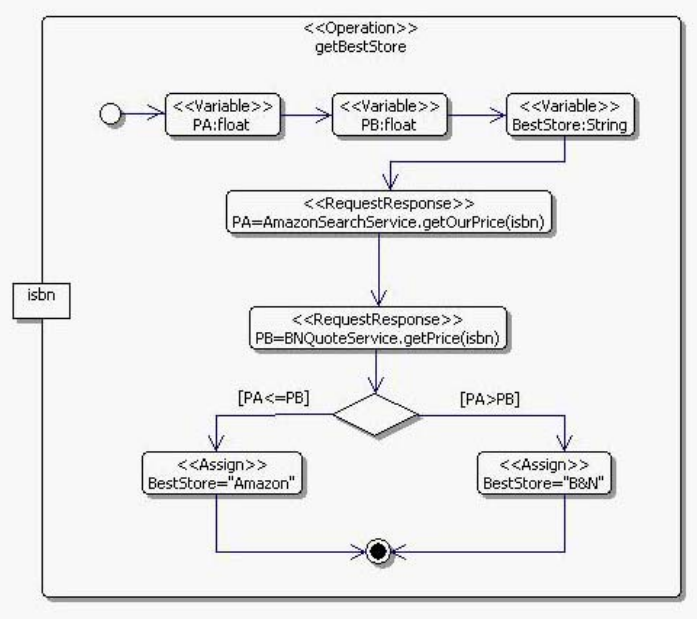


Figura 4.15 MDCS para la operación `getBestStore`

El MDCS se define en una actividad estereotipada con la palabra clave `<<Operation>>`. Esta actividad recibe como entrada el `isbn` del libro (modelado como un *input pin* de la actividad) Al inicio se declaran algunas variables locales a la operación. Posteriormente se invocan a operaciones de los servicios ajenos

Amazon y B&N para la consulta de precios y finalmente se hace una comparación de los resultados obtenidos para determinar cual es la mejor tienda.

4.2.3.3 Selección dinámica de servicio

La selección dinámica de servicios se especifica en el MDCS mediante el uso combinado de dos acciones: (1) una declaración de variable de tipo *Service* y (2) una acción *SelectService*, en la cual se inicializa la variable anterior invocando la operación *getXService* (definida como parte de la fachada del servicio dinámico en el Modelo de Servicios –ver Figura 4.13-). Esta operación se ejecuta con el propósito de seleccionar dinámicamente el servicio en base a una condición establecida por el modelador, mediante una expresión OCL[12] en el Modelo de Servicios.

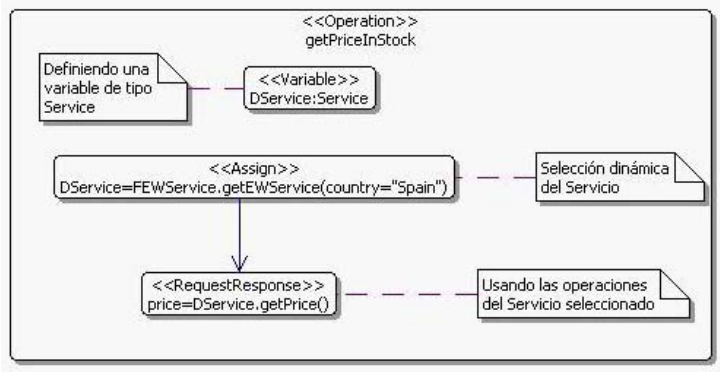


Figura 4.16 Selección dinámica de servicios en el MDCS

La Figura 4.16 muestra un ejemplo de selección dinámica de servicio Web para el ejemplo de los almacenes centrales de la Figura 4.12. Con el fin de implementar la operación `SService.getPriceInStock` se debe definir una variable tipo *Service* y después se debe combinar las dos acciones mencionadas. Para este ejemplo la condición se define sobre la variable `country` del MDCS, enviada como parámetro a la condición de la operación definida en la fachada.

[12] Que pudiera utilizar las variables del MDCS (variables del proceso).

4.2.4 Gestión del modelo: el paquete *Management*

El paquete *Management* (ver Figura 4.17) incluye las metaclases necesarias para la organización arquitectónica de los servicios. La primitiva arquitectónica básica es *Subsystem* que permite definir un conjunto cohesivo de servicios. A su vez cada *Subsystem* puede agrupar cero o más *Subsystems* en una relación jerárquica. El Modelo de Servicios se construye a partir de uno o más *Subsystems*.

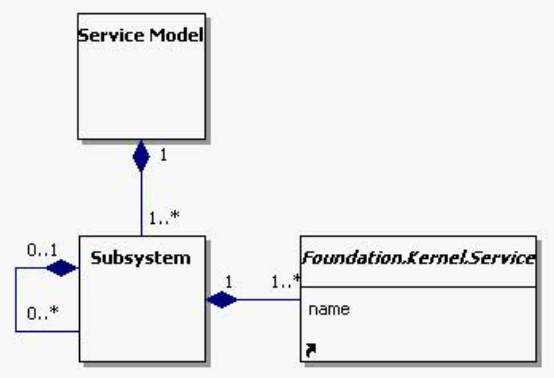


Figura 4.17 El paquete Management

4.3 Conclusiones

El metamodelo de servicios expuesto se agrega a los metamodelos existentes en el método OOWS para especificar los aspectos relativos a la producción y consumo de funcionalidad de la aplicación Web. Este metamodelo de servicios ha sido definido con un conjunto de propiedades y restricciones que permiten definir herramientas que posibiliten la verificación de los modelos. Una vez definido este metamodelo de servicios el siguiente paso es su integración con los modelos existentes en OOWS, los cuales requerirán de algunas adaptaciones para lograr esta integración. La integración y adaptaciones necesarias son explicadas en el capítulo siguiente de la tesis.

CAPÍTULO 5

Modelado de aplicaciones Web basadas en servicios Web

El presente capítulo expone el conjunto de adecuaciones y extensiones al método OOWS, que permiten el modelado conceptual de aplicaciones Web con capacidad de integración mediante servicios Web. Partiendo de la definición original del método, este capítulo propone un conjunto de adecuaciones y extensiones a sus modelos para lograr la especificación de aplicaciones Web basadas en servicios Web, capaces de consultar y actualizar datos, así como de producir y consumir funcionalidad mediante servicios Web.

El capítulo se organiza en tres partes: en la primera se discuten las razones y motivos que justifican las adecuaciones y extensiones y luego ofrece la redefinición del método. En la segunda, se introduce la redefinición del Modelo Navegacional y su integración con la notación de los Modelos de Servicios y Dinámico de Composición de Servicios del capítulo anterior. En la tercera, se explica la extensión al Modelo de Presentación para las operaciones de servicios Web que ofrecen resultados. Los aspectos discutidos son explicados mediante un ejemplo de aplicación Web para el catálogo de biblioteca de la UPV.

5.1 Redefinición del enfoque metodológico y adecuación de modelos

La base de las extensiones y adecuaciones se encuentra en la redefinición del enfoque metodológico de OOWS. Las premisas de la redefinición están dadas en los siguientes apartados.

5.1.1 Premisas de la redefinición metodológica

Las adecuaciones y extensiones necesarias al método OOWS para especificar aplicaciones Web que integran datos y funcionalidad a partir de servicios Web se fundamentarán en las siguientes premisas: (1) *respeto al principio de separación de aspectos*, para lo cual se propone la captura de las nuevas características en modelos diferenciados; (2) *integración de estos nuevos modelos con los ya existentes*, adecuando y extendiendo sus primitivas conceptuales, así como definiendo nuevas cuando sea necesario, (3) *inclusión de modelos de procesos Negocio-a-Negocio*, a partir de los cuales sea posible la definición de una aplicación Web que les de soporte (4) *definición de métodos para su aplicación y uso* e (5) *introducción de nuevas transformaciones de modelos* para la generación del código que implementa la funcionalidad de integración de la aplicación, así como el enlace con los componentes software generados a partir de los modelos ya existentes. Todo lo anterior enmarcado en el contexto de un método en el cual se redefine el Modelo Navegacional integrándolo a los Modelos de Servicios y Dinámico de Composición de Servicios, introducidos en el capítulo anterior.

Las premisas (1) y (2) son la base de los aspectos discutidos en el presente capítulo. Las premisas (3) y (4) son la base de lo discutido en el capítulo 6. La premisa (5) fundamenta lo que se presentará en el capítulo 7.

5.1.2 Redefinición del método

El método OOWS se redefine incluyendo nuevos pasos y cambiando algunos de los ya existentes (Figura 5.1). La redefinición incluye también la introducción de los nuevos modelos propuestos en el capítulo anterior, su integración con los modelos existentes, adecuación de algunos de éstos, así como nuevas reglas de transformación de modelos a código final.

La primera fase (*Especificación del Sistema*) agrega un paso de Modelado de integración y considera los pasos de Modelado de navegación y presentación de la siguiente manera:

- **Modelado de Integración:** (caja 4 en Figura 5.1), donde se especifican los aspectos de integración de la aplicación mediante el *Modelo de Servicios* y el *Modelo Dinámico de Composición de Servicios.* En el primero se especifica la funcionalidad que produce (mediante servicios propios) y consume (mediante servicios ajenos) la aplicación. La funcionalidad que produce puede ser definida a partir de vistas del Modelo Estructural o por composición de servicios. En el segundo se especifica la composición de las operaciones de servicios compuestos.
- **Modelado de Navegación y Presentación:** (caja 3 en Figura 5.1), en este paso se redefinen las clases navegacionales en dos tipos de *Unidades de Interacción*: de *Información* y *Funcionalidad*. Ambas se establecen como vistas de los Modelos Estructural y de Servicios. Estas Unidades de Interacción se enlazan mediante un conjunto de relaciones para especificar un espacio de navegación hipermedial que consulta y modifica datos locales, consulta datos a partir de funcionalidad propia o ajena e invoca funcionalidad local y de los servicios propios y ajenos. Finalmente los requisitos de presentación se capturan en el *Modelo de Presentación*.

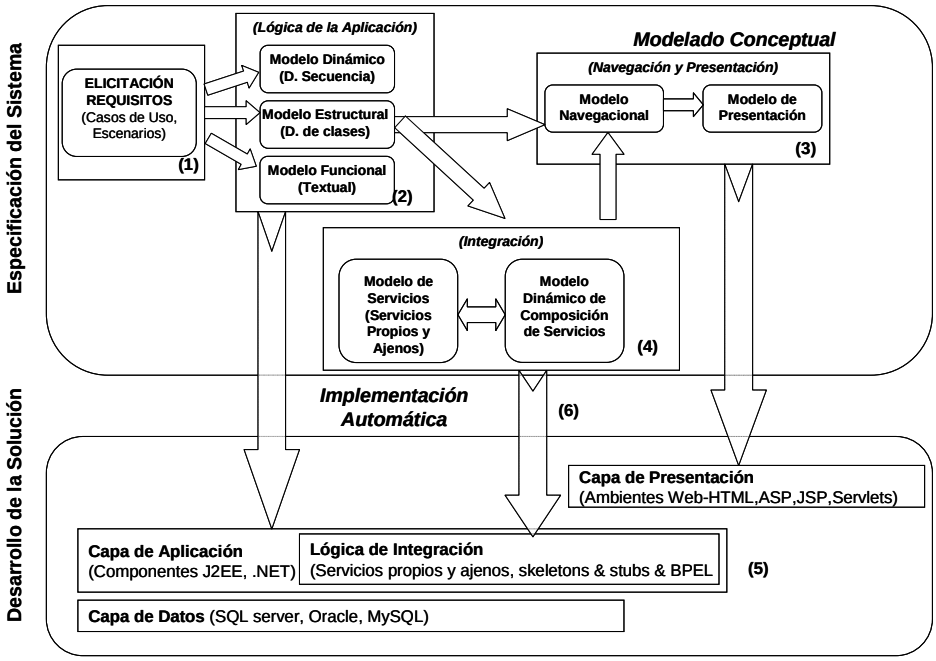


Figura 5.1 Redefinición del enfoque metodológico OOWS

En la segunda fase (*Desarrollo de la Solución*, caja 5 en Figura 5.1), se adecua la arquitectura software en su capa de Aplicación para incluir una subcapa de *Lógica de integración*. Esta subcapa posee todos los componentes necesarios para la implementación de las capacidades de integración de la aplicación.

Las correspondencias (6 en Figura 5.1) entre los modelos conceptuales y sus representaciones software también son extendidas. Considerando ahora la presencia de lógica de integración, tanto el *Modelo de Servicios* como el *Modelo Dinámico de Composición de Servicios* se transforman en componentes del subsistema de *Lógica de Integración* para el consumo[13] y producción[14] de servicios Web. Se genera también un conjunto de fachadas para los servicios Web propios y ajenos que facilitan su acoplamiento con los componentes restantes, principalmente los de la capa de Presentación. En el caso de operaciones de servicios compuestos, las corresponden-

[13] Mediante un tipo de componente software intermedio que posee la lógica de comunicación necesaria para que la aplicación Web consuma servicios Web ajenos. Son llamados en inglés *stubs*.

[14] Mediante un tipo de componente software intermedio que es utilizado por aplicaciones externas para la comunicación con los servicios Web propios. Son llamados en inglés *skeletons*.

cias generan el código para la implementación de la composición de las operaciones, incluyendo lo necesario para su ejecución e instalación en motores de ejecución de servicios compuestos (tipo WS-BPEL).

En los siguientes apartados se explican estas adecuaciones al método y sus modelos. La explicación se apoya en un caso de estudio sobre el catálogo de biblioteca la Universidad Politécnica de Valencia (`http://www.upv.es/bib/`). Todas las transformaciones de modelos a modelos y de modelos a código se han organizado en cuatro escenarios que se explicarán en el capítulo 7.

5.1.3 Caso de estudio: el catálogo de biblioteca de la UPV

El catálogo de biblioteca de la UPV posee información sobre los libros y audiovisuales existentes en las bases de datos de las bibliotecas locales de sus diversos departamentos, así como de la biblioteca central. Esta información es consultada mediante interfaces Web. Un ejemplo de página Web de consulta está dado en la Figura 5.2.

Figura 5.2 Página Web del catálogo de biblioteca de la UPV

En los pasos iniciales del método, se identifican los usuarios de la aplicación y se construye para cada uno de ellos el *Mapa Navegacional* que representa su vista particular de la aplicación Web. Esta tarea se realiza en los pasos de *Autoreo-a-escala-mayor* y *Autoreo-a-escala-menor* que se detallan a continuación.

5.1.3.1 Autoreo-a-escala-mayor

La Figura 5.3 muestra un extracto del *Mapa Navegacional* para el usuario `Anónimo` del catálogo de biblioteca de la UPV.

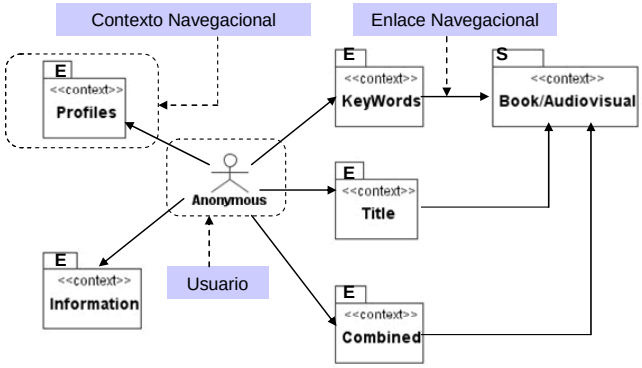


Figura 5.3 Mapa Navegacional para el usuario `Anónimo`

La estrategia original de definición del Mapa Navegacional no cambia en la redefinición: el usuario `Anónimo` tiene acceso directo y desde cualquier punto a los contextos navegacionales de Exploración `Profiles`, `Information`, `Keywords`, `Title` y `Combined`. En contraparte, sólo puede alcanzar el contexto navegacional de Secuencia `Book/Audiovisual` a través de los enlaces navegacionales que parten de los contextos `Keywords`, `Title` o `Combined`.

Una vez que el Mapa Navegacional ha sido definido, en el siguiente paso, *Autoreo-a-escala-menor*, se detalla cada contexto navegacional. Esta etapa se realiza considerando las adecuaciones y agregados que se discuten a continuación.

5.1.3.2 Autoreo-a-escala-menor

En esta fase se diseñarán cada uno de los contextos navegacionales de tal manera que ofrezcan al usuario la posibilidad de *consultar* información e *invocar* funcionalidad. Estas tareas se especificarán en los contextos navegacionales mediante alguno de los tipos de *Unidades de interacción*: de *Información* o *Funcionalidad*. Para los requisitos de consulta de datos e invocación de funcionalidad local, no

implementada como servicios Web, se define la *Unidad de Información (IU)*[15]. Para los requisitos de consulta de datos e invocación de funcionalidad, implementados mediante servicios Web, se define la *Unidad de Funcionalidad (FU)*[16].

La Figura 5.4 muestra el contexto navegacional `Book/Audiovisual` para la página Web de la Figura 5.2. Este contexto navegacional incluye únicamente datos locales, por lo que se diseña utilizando solamente IUs.

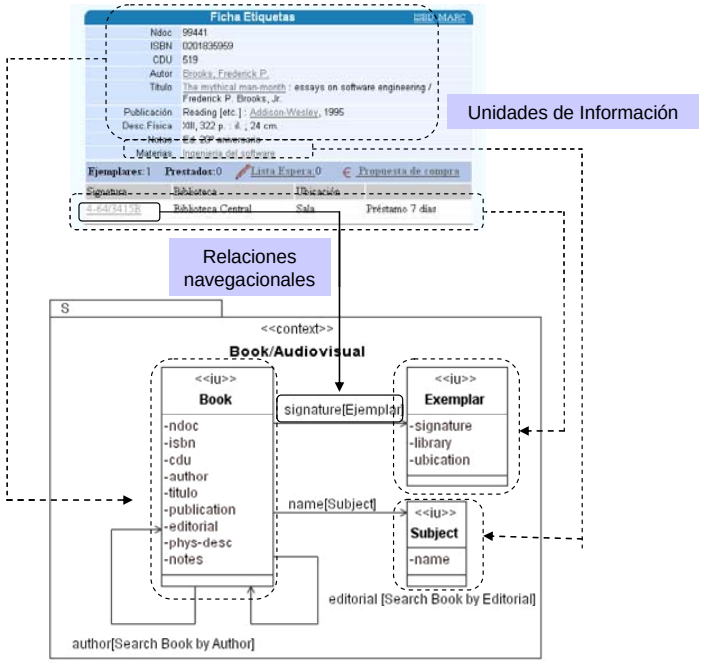


Figura 5.4 El contexto navegacional `Book/Audivisual`

Las IUs `Book`, `Exemplar` y `Subject`, se definen como vistas de las clases `Book`, `Exemplar` y `Subject` del *Diagrama de clases*. Las IUs están enlazadas mediante *relaciones navegacionales* contextuales, acompañadas de *atributos de contexto*, que representan rutas de navegación hacia otros contextos: `Book-Exemplar`, para navegar hacia el contexto `[Ejemplar]`; `Book-Subject`, para navegar hacia el contexto `[Subject]` y `Book-Book`, para navegar hacia el contexto `[Search Book by Author]`. Cada relación posee también un *atributo de enlace* que representa el dato de la página actual que se utilizará para el enlace al contexto destino: por ejemplo en la relación `Book-Exemplar` el *atribu-*

[15] Se utilizarán de aquí en adelante las siglas en inglés de *Information Unit* (IU).
[16] Se utilizarán de aquí en adelante las siglas en inglés de *Functionality Unit* (FU).

to de enlace está dado por `signatura`; en la relación `Book-Subject` se utiliza el atributo `name` y en `Book-Book` se utiliza `author`.

Agregación de datos a partir de servicios Web

Mediante FUs, se pueden diseñar contextos navegacionales que incluyan datos y funcionalidad definidos a partir de las operaciones de los servicios Web propios o ajenos del Modelo de Servicios. Por ejemplo, el contexto `Book/Audiovisual` se puede enriquecer con datos provenientes de los servicios Web ajenos `Amazon`[47], `Barnes&Noble`[48] y `Google` [82,83] para determinar, entre las tiendas `Amazon` y `Barnes&Noble`, el menor coste de un ejemplar; consultar y mostrar la imagen del ejemplar a partir del banco de imágenes de `Amazon`, así como consultar y mostrar precios del mismo ejemplar en otras librerías, utilizando `Google`. Estos requisitos se pueden realizar incluyendo la FU *InfoWeb* (Figura 5.5) en el contexto navegacional.

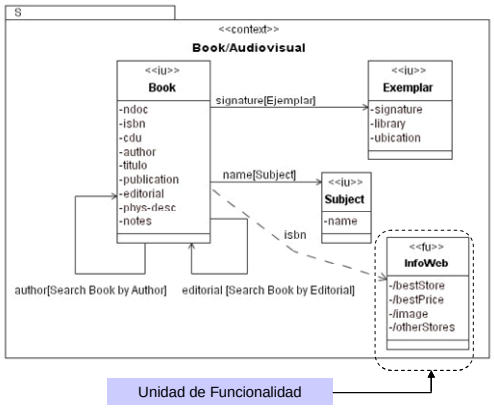
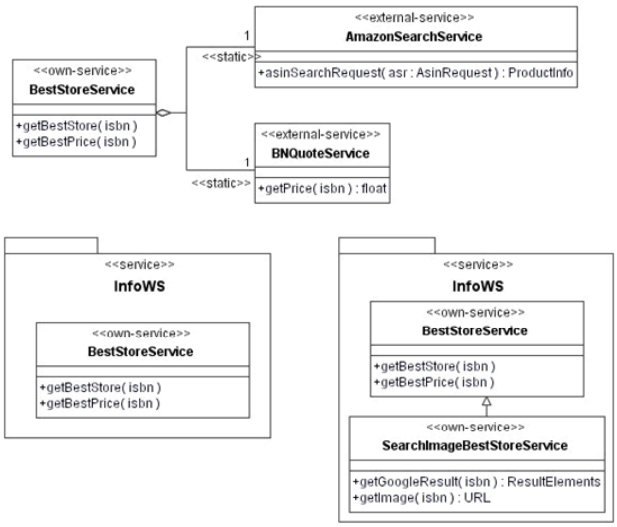


Figura 5.5 El contexto navegacional `Books/Audiovisual` utilizando una FU

Cada uno de los atributos de la FU es un atributo derivado que se define en función de las operaciones del servicio Web propio *InfoWS*, diseñado a partir de los servicios Web ajenos `Amazon`, `B&N` y `Google`, importados previamente al Modelo de Servicios. El Modelo de Servicios y la definición de atributos están dados en la Figura 5.6.




```
bestStore=InfoWS.BestStoreService.getBestStore(isbn)
bestPrice=InfoWS.BestStoreService.getBestPrices(isbn)
image=InfoWS.SearchImageBestStoreService.getImage(isbn)
otherStores= InfoWS.SearchImageBestStoreService.getGoogleResult(isbn)
```

Figura 5.6 Modelo de Servicios y definición de atributos de una FU

La FU también requiere del *atributo de entrada* `isbn`, que transporta dicho dato desde la IU `Book` hacia la FU. Este atributo se utiliza en la definición de cada uno de los atributos de la FU.

La definición de las primitivas conceptuales utilizadas para el modelado navegacional, han sido diseñadas para mantener compatibilidad con las primitivas originales OOWS agregando nuevas para la especificación de los nuevos requisitos implicados por la presencia de los servicios Web.

Cada una de las operaciones del servicio propio `BestStoreService` se especifica en un *Modelo Dinámico de Composición de Servicios*. Por ejemplo, la Figura 5.7 muestra la definición para la operación `getBestStore`.

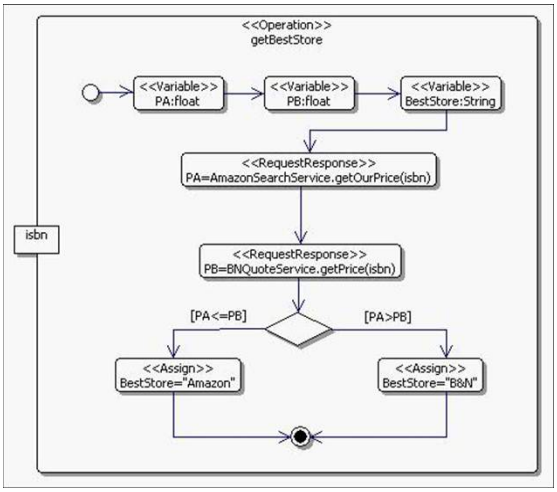


Figura 5.7 MDCS para la operación `getBestStore`

Modelado de presentación

Los requisitos de presentación se capturan en el *Modelo de Presentación*. Este modelo se relaciona con el Modelo Navegacional y define, para cada una de sus primitivas, algún patrón de presentación. La Figura 5.8 muestra el Modelo de Presentación para el contexto `Book/Audiovisual` en el cual se establece que la IU directora `Book` se muestre aplicando el patrón *Registro* y su información relacionada (IUs `Exemplar`, `Subject`, `InfoWeb`) se muestre con el patrón *Maestro-Detalle*. El detalle se mostrará a su vez aplicando el patrón *Registro*. Este Modelo de Presentación mantiene compatibilidad con la definición original del método OOWS, sólo que ahora, además, especifica propiedades de presentación para los atributos de la FU. Otro requisito que es necesario capturar está dado por la presentación de resultados obtenidos a partir de operaciones de servicios Web en la FU. Se explica más adelante.

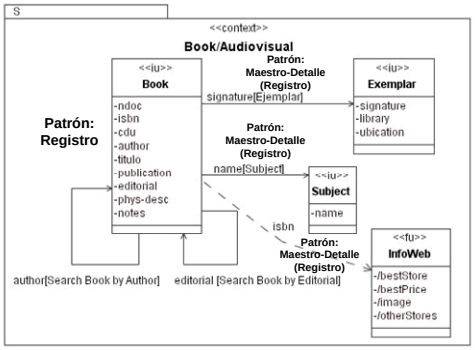


Figura 5.8 Modelo de Presentación incluyendo IUs y FU

Con las extensiones mostradas la página Web se enriquece ahora con información derivada de servicios Web, tal como lo muestra la Figura 5.9.

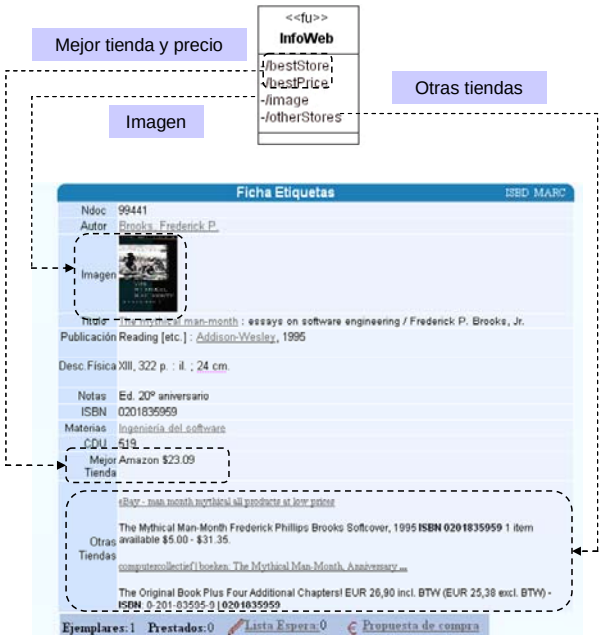


Figura 5.9 Página Web enriquecida con información derivada de servicios Web

Las nuevas unidades de interacción, utilizadas en el ejemplo, se introducen al Modelo Navegacional modificando su metamodelo MOF presentado en la siguiente sección.

5.2 Redefinición del contexto navegacional

El metamodelo MOF (Figura 5.10) del Contexto navegacional (*Navigational Context*) incluye ahora la metaclase Unidad de Interacción (*Interaction Unit*) en lugar de la metaclase Clase navegacional. La *Unidad de Interacción* ofrece un trato uniforme a las vistas de datos y funcionalidad para las clases de los Modelos Estructural y de Servicios.

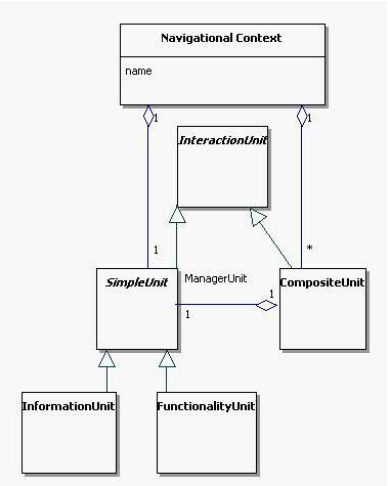


Figura 5.10 El metamodelo redefinido para el contexto navegacional

Cada *Unidad de Interacción* podrá ser: (1) *Simple*: *Unidad de Información* o *de Funcionalidad* o (2) *Compuesta*: para especificar agregación de contenidos (semejante a las AIUs). A continuación se presentan los metamodelos para los dos tipos de Unidades de Interacción.

5.2.1 La Unidad de Información

Una *Unidad de Información* (IU) (Figura 5.11) es una *Unidad de Interacción Simple* definida como una *vista* de los atributos y operaciones de una clase del *Diagrama de clases*. Es una unidad cohesiva que permite la consulta de datos e invocación de funcionalidad derivada del *Modelo Estructural*. La IU se representa mediante una clase UML estereotipada con la palabra clave `<<IU>>`.

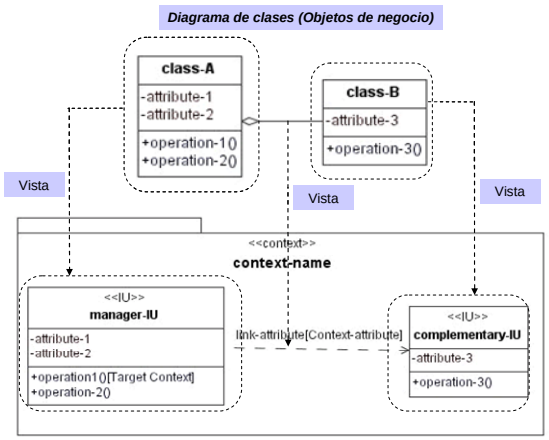


Figura 5.11 La Unidad de Información como vista del Diagrama de clases

La Figura 5.12 muestra el metamodelo MOF de la IU. El Contexto navegacional (*Navigational Context*) contendrá una *IU directora* (*Manager Unit*) para especificar la información principal del mismo y un conjunto (posiblemente vacío) de *IU complementarias* (*Complementary Unit*). Ambas se enlazarán mediante *relaciones navegacionales* (*Navigational Relationship*) que se definen sobre las relaciones de asociación, agregación o especialización del *Diagrama de clases*. Estas relaciones podrán ser de dos tipos posibles: (1) *de dependencia contextual* (*Contextual Dependency Relationship*), representando recuperación de información relacionada entre las IUs o (2) *de contexto* (*Context Relationship*), las cuales además de recuperación de información, proveen capacidad de navegación a un contexto destino. La *relación navegacional de contexto* se especifica mediante dos propiedades: un *atributo de contexto* (*Context Attribute*), para identificar el contexto destino y un *atributo de enlace* (*Link Attribute*), para indicar el atributo que será utilizado en la página Web para señalar la liga a una IU destino.

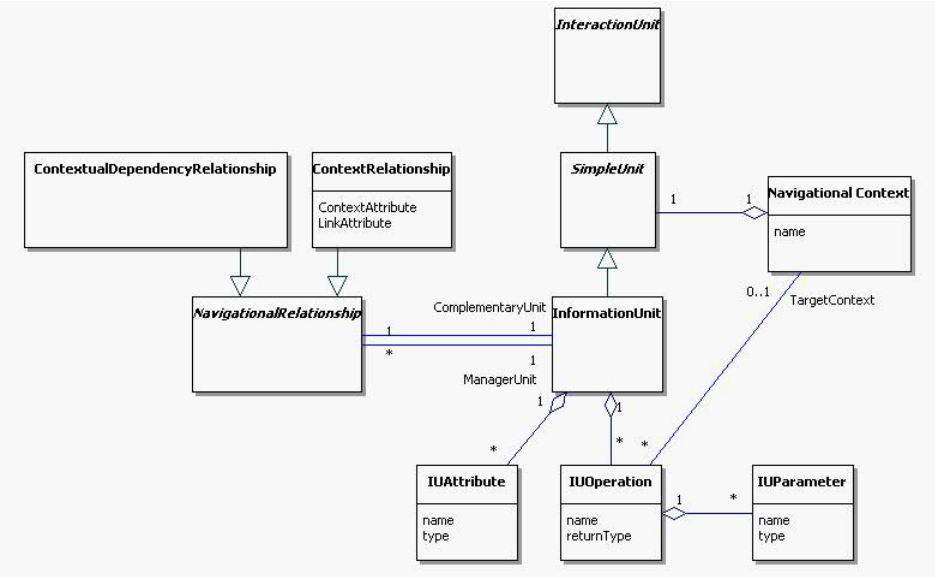


Figura 5.12 El metamodelo de la Unidad de Información

La IU mantiene compatibilidad con la anterior definición de Clase navegacional, con lo cual se facilita el proceso de aprendizaje del diseñador, familiarizado con la definición anterior del Modelo Navegacional.

5.2.1.1 La IU como vista del resultado de una operación de servicio Web

Una excepción de la IU, como vista del *Diagrama de Clases*, es su definición como vista de la respuesta de una operación de servicio Web *Request-response*. Esta posibilidad permite incluir en un contexto navegacional la respuesta como datos obtenidos a partir de la invocación de la operación. En este caso la IU se define a partir de uno o más atributos de su valor de retorno (*Return Value*).

La definición de la llamada a la operación del servicio Web está dada en mediante un mecanismo de **Consulta de Servicio Web (WS-Query)**, establecido en el último compartimento de la IU (ver Figura 5.13). Este mecanismo requiere que el diseñador establezca valores para las siguientes cláusulas:

- **WS-QUERY**: el identificador de la consulta.
- **OPERATION**: el nombre de la operación del servico Web a invocar.
- **PARAMETERS**: los parámetros de la operación.

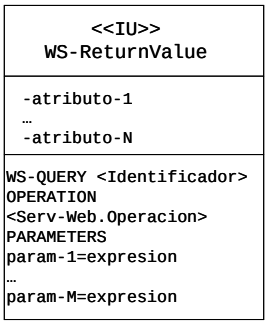


Figura 5.13 La Consulta de Servicio Web en una IU

Mediante este tipo de IU es posible, entonces, mostrar los resultados de una invocación de una operación de forma semejante a como se consultan datos a partir del *Diagrama de Clases*. La Figura 5.14 muestra la IU mostrando los resultados de una la búsqueda del tema "*Web Engineering*" en Google.

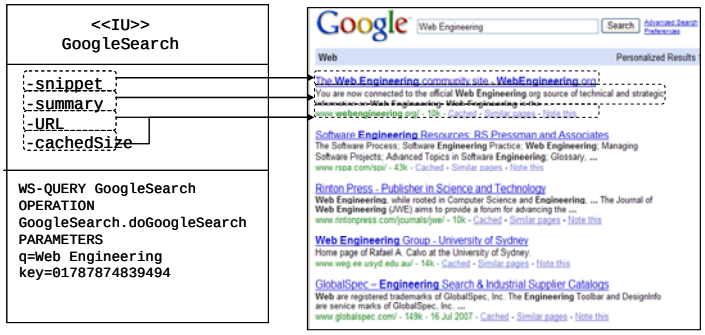


Figura 5.14 IU para una búsqueda en el servicio Web `GoogleSearch`

5.2.1.2 El Modelo de Presentación para la IU

El *Modelo de Presentación* para la IU mantiene compatibilidad con el Modelo de Presentación anterior para las clases navegacionales. Se aplican los mismos patrones, por lo que no se explica de nuevo aquí.

5.2.2 La Unidad de Funcionalidad

Una *Unidad de Funcionalidad* (FU) (Figura 5.15) es una *Unidad de Interacción Simple* definida a partir del Modelo de Servicios. Es una unidad cohesiva de datos y funcionalidad que se deriva a partir de dicho modelo. Los datos se definen

con una expresión definida en función de las operaciones del modelo. La funcionalidad se define a partir de vistas de las operaciones de los servicios propios o ajenos del modelo.

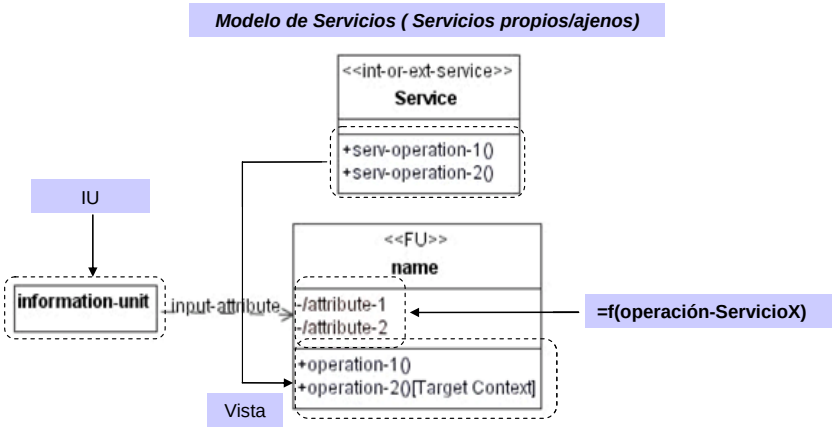


Figura 5.15 La Unidad de Funcionalidad

La FU se representa mediante una clase UML estereotipada con la palabra clave <<FU>>. Los atributos se expresan mediante la notación UML de atributo derivado y su definición está dada por una regla de derivación OCL.

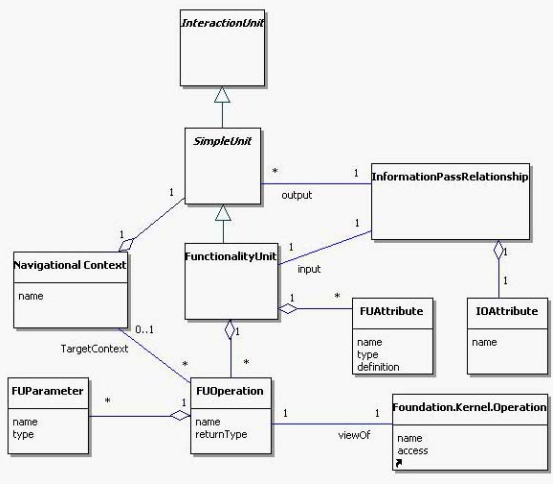


Figura 5.16 El metamodelo de la Unidad de Funcionalidad

La Figura 5.16 muestra el metamodelo de la FU. Sus operaciones (*FUOperation*) permiten la invocación de las operaciones del Modelo de Servicios desde la página Web. Existen dos tipos de operaciones: (1) operaciones con capacidad de navegación a un contexto destino y (2) operaciones sin capacidad de navegación. Para las primeras, las cuales se expresan con una notación similar a la IU, (`operation-2` en Figura 5.15) la respuesta se mostrará en el contexto actual dependiendo del tipo de operación: si la operación tiene respuesta (*request-response* o *solicit-response*), ésta se muestra en el contexto actual sin transferir la navegación a un contexto diferente; si la operación no tiene respuesta (*one-way* o *notify*), la navegación se transferirá a un contexto destino sin mostrar ninguna respuesta.

Es posible enviar información contextual desde una IU a una FU mediante uno o más *atributos de entrada* (*IOAttribute*). Los valores de estos atributos pueden ser utilizados por las operaciones de la FU como parámetros (no definidos explícitamente) en el encabezado de la operación. Pueden también ser utilizados en la regla de derivación OCL para los atributos de la FU.

5.2.2.1 El Modelo de Presentación para operaciones en FU

El *Modelo de Presentación* se extiende también para definir los requisitos de presentación de la aplicación Web en las operaciones de la FU que ofrecen respuesta (*request-respose* o *solicite-response*). La definición se basa en las siguientes propiedades y patrones de presentación asociados a cada operación de este tipo:

- **Paginación de resultados:** se aplica cuando el resultado de la operación es una colección. Los resultados se dividen en bloques, de los cuales se visualiza uno a la vez. Ofrece mecanismos de paginación para avanzar y retroceder, así como acceso indexado a cualquiera de los bloques. Para su definición requiere información de:
  - **Cardinalidad**: para indicar el número de resultados a mostrar por bloque.
  - **Criterio de ordenamiento**: para definir si los resultados se mostrarán en orden ascendente o descendente, de acuerdo a algún atributo del resultado. Es opcional, por defecto es sin ningún ordenamiento.

- **Atributo de ordenamiento**: para especificar el o los atributos del resultado por los que se mostrará ordenada la información. Se debe proveer si se selecciona un criterio de ordenamiento.

- **Distribución de la información**: define la forma en la que serán mostrados los resultados en el contexto. Incluye tres patrones: Registro (*Register*), Tabular (*Tabular*) o Renglón (*Row*) (formato libre).

Estas propiedades y patrones se suman a los ya existentes en el Modelo de Presentación para las IU. Para las propiedades de presentación de los atributos de la FU, se aplican estos mismos patrones. Los patrones de presentación para operaciones con resultados en la FU se especifican en el último compartimiento (Figura 5.17).

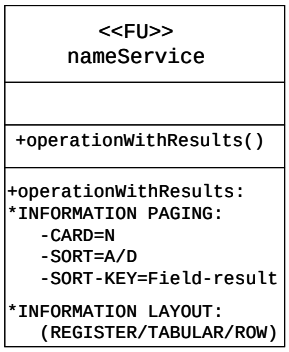


Figura 5.17 Propiedaes y patrones de presentación para operaciones con respuesta en la FU

La Figura 5.18 muestra un ejemplo de *Modelo de Presentación* para la operación `doGoogleSearchService` del servicio Web de `Google`, una operación *request-response*, que devuelve una colección de los resultados de la búsqueda. En el modelo se indica que se mostrarán cinco resultados, ordenados por el criterio de importancia (*rate*). Al final se ofrecen mecanismos de navegación a los bloques de resultados.

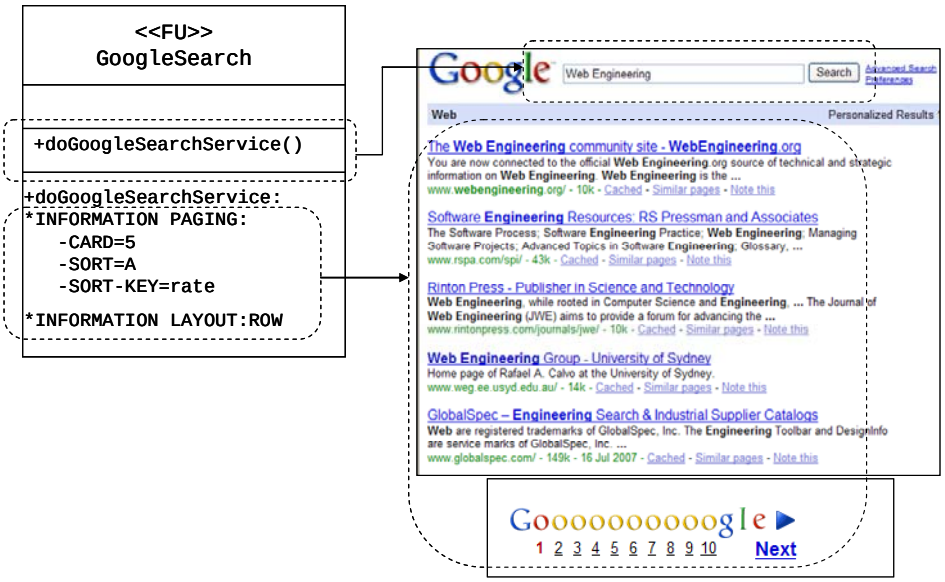


Figura 5.18 El Modelo de Presentación para una búsqueda en Google

5.3 Conclusiones

Se han presentado las adecuaciones y extensiones al método OOWS y a sus modelos de navegación y presentación para la especificación de aplicaciones que hacen uso de servicios Web. Estas adecuaciones y extensiones se han diseñado manteniendo compatibilidad con la versión anterior del método con el fin de facilitar el proceso de aprendizaje y transición de los diseñadores, así como también se ha buscado respetar las premisas (1) *principio de separación de aspectos* e (2) *integración con los nuevos modelos* en la solución ofrecida. El siguiente capítulo aborda un aspecto complementario en este tipo de aplicaciones: la obtención de aplicaciones Web a partir de modelos de procesos Negocio-a-Negocio.

CAPÍTULO 6

Transformación de Procesos Negocio-a-Negocio a Modelos navegacionales

En el presente capítulo se aborda el problema de obtención de modelos navegacionales para aplicaciones Web basadas en servicios, a partir de procesos Negocio-a-Negocio. El capítulo ofrece un método para la obtención del Mapa Navegacional OOWS partiendo de la definición de un proceso caracterizado por la colaboración entre actores humanos y aplicaciones externas. Este es un aspecto complementario de la propuesta de inclusión de servicios Web al método OOWS ofrecido en la tesis.

El capítulo se organiza de la siguiente manera: en la primera parte se motiva el tema y se ofrecen razones por las cuales es conveniente abordar el problema. En la segunda, se introduce el método haciendo uso del Metamodelo de la Especificación de Procesos de Ingeniería de Software. En la tercera y cuarta parte, se explica la estrategia, apoyándose en un caso de estudio para el desarrollo de la aplicación Web `Merkalink.com`. La tercera parte, explica las dos primeras disciplinas del método, introduciendo modelos y técnicas; y en la cuarta, se explica la última disciplina, introduciendo reglas de transformación de modelos para la obtención del Mapa Navegacional. Finalmente, la última sección ofrece conclusiones.

6.1 Motivación

Es cada vez mayor el papel que la Internet está jugando como plataforma para la implementación de procesos Negocio-a-Negocio, en los cuales se requiere la participación humana a través de interfaces Web así como la colaboración con aplicaciones externas.

Este tipo de procesos pueden ser especificados mediante modelos que sean la base para la construcción de aplicaciones Web que soporten la interacción humana, así como la integración con aplicaciones externas. La construcción de estos modelos sugiere un acercamiento entre los modelos de procesos Negocio-a-Negocio y los métodos de Ingeniería Web. La propuesta de esta tesis es obtener la aplicación Web mediante un método de transformación de modelos que en su entrada reciba al modelo de procesos y en su salida obtenga el Mapa Navegacional.

Esta estrategia se ofrece en este capítulo para el método OOWS. Iniciando con un proceso Negocio-a-Negocio se aplican pasos de transformación hasta la obtención del Mapa Navegacional (basado en la redefinición de primitivas introducida en el capítulo anterior) dando soporte a la interacción humana y a la integración de aplicaciones externas mediante servicios Web.

6.2 El método

En esta sección se introduce el método para la obtención del Mapa Navegacional. Siguiendo un enfoque semiautomático, de arriba hacia abajo, se establece trazabilidad desdesde la definición del proceso Negocio-a-Negocio hasta el Mapa Navegacional. Para su explicación se utiliza la notación y términos del Metamodelo para la Especificación de Procesos de Ingeniería de Software[17] (EPIS) [84].

En términos de EPIS el método es llamado *Proceso* (de aquí en adelante también llamado *Proceso top-down*). Cada Proceso se define en términos de *Disciplinas*, un conjunto cohesivo de *Actividades*[18] realizadas por un *Rol del proceso*[19] que

[17] En ingles *Software Process Engineering Metamodel Specification* (SPEM)
[18] *Activity*, término en inglés usado en SPEM
[19] *Process role*, término en inglés usado en SPEM

obtienen un *Producto de trabajo*[20]. La Figura 6.1 muestra el Proceso propuesto y sus tres disciplinas: *Requisitos, Diseño* y *Generación de código*. Se explican a continuación.

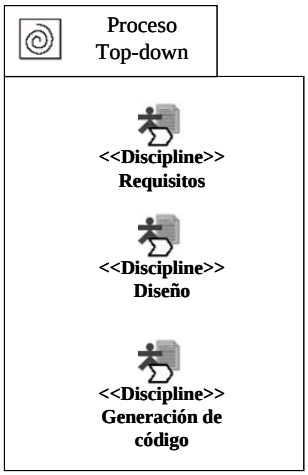


Figura 6.1 El proceso Top-down

6.2.1 Disciplina de Requisitos

La Figura 6.2 muestra el Diagrama de Paquetes para la disciplina de *Requisitos*.

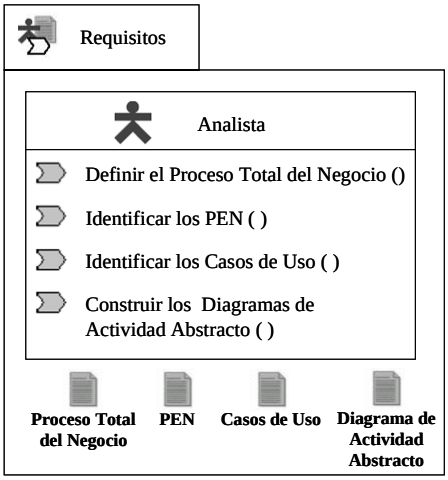


Figura 6.2 La disciplina de Requisitos

[20] *Work Product*, término en inglés usado en SPEM

En esta disciplina el *Analista* efectúa tres actividades principales:

1. Establece el modelo general del negocio, incluyendo todos los procesos y participantes del mismo. A este modelo le llamará el *Proceso Total del Negocio*.
2. A partir del *Proceso Total del Negocio*, identifica los *Procesos Elementales del Negocio* (PEN) [51] los cuales se corresponderán con *Casos de Uso*.
3. Para cada *Caso de Uso*, construye un *Diagrama de Actividad Abstracto* que incluirá los actores y sus actividades. Los actores serán: (1) usuarios que interactúan directamente con el sistema, (2) la aplicación Web, como sistema que realiza acciones a partir de la interacción del usuario y (3) aplicaciones externas con las que se comunica la aplicación Web.

El *Diagrama de Actividad Abstracto* es el Producto de trabajo final de esta disciplina y es también la entrada a la siguiente disciplina de *Diseño*.

6.2.2 Disciplina de Diseño

Tomando en su entrada el *Diagrama de Actividad Abstracto*, en esta disciplina (Figura 6.3) el Diseñador realiza las siguientes actividades:

1. Partiendo del *Diagramas de Actividad Abstracto* obtiene el *Diagrama de Clases*, incluyendo los conceptos del dominio.
2. El *Diagrama de Actividad Abstracto* también lo utiliza para definir el *Modelo de Servicios* incluyendo los servicios y operaciones necesarias para lograr la integración de la aplicación actual con las aplicaciones externas.
3. Considerando los *Diagramas de Actividad Abstracto*, *Diagrama de Clases* y el *Modelo de Servicios*, refina el primero en un *Diagrama de Actividad Concreto* que corresponderá a la aplicación actual para el *Proceso Total del Negocio*.

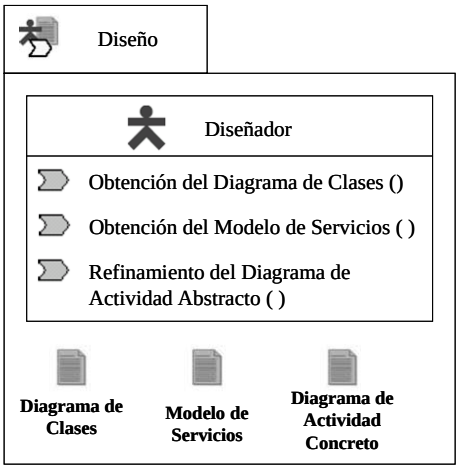


Figura 6.3 La disciplina de Diseño

El *Diagrama de Actividad Concreto* será el Producto de trabajo en la salida de esta disciplina, el cual, a su vez, será la entrada para la última disciplina de *Generación de código*.

6.2.3 Disciplina de Generación de Código

El propósito de esta disciplina (Figura 6.4) es la obtención del *Mapa Navegacional* a partir del *Diagrama de Actividad Concreto*. Mediante un conjunto de reglas de transformación, se identifican patrones de relación entre actividades los cuales se corresponden con la estructura navegacional del mapa. Una vez obtenido éste, se aplica la estrategia de generación de código de OOWS para obtener la aplicación Web final.

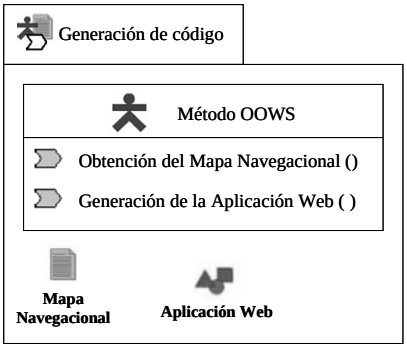


Figura 6.4 La disciplina de Generación de código

6.2.4 Secuencia de actividades

La secuencia completa de las actividades del método se muestra en la Figura 6.5. La secuencia incluye en un solo diagrama: *Roles del proceso*, *Actividades* y *Productos de trabajo*.

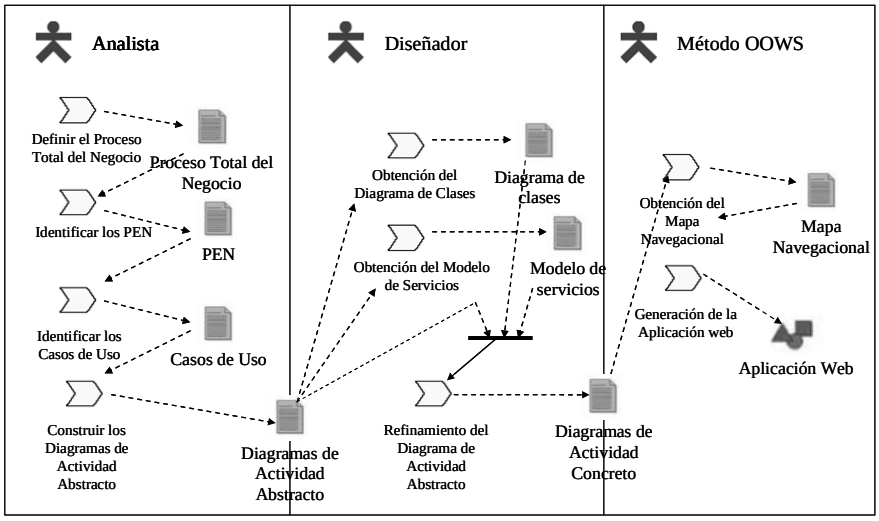


Figura 6.5 Secuencia de actividades para el Proceso Top-down

6.3 Un caso de estudio: Merkalink.com

El método propuesto se ilustra a continuación mediante el desarrollo de la aplicación de comercio electrónico `Merkalink.com`: un sitio Web diseñado para construir un enlace entre los mercados en línea de EEUU y Canadá con México.

El Modelo de Negocio básico es el siguiente: usando una dirección postal en Laredo, Texas (en los EEUU) los usuarios en México pueden comprar artículos en las tiendas en-línea de los EEUU o Canadá y solicitar su envío a dicha dirección. Posteriormente `Merkalink.com` envía el artículo desde Laredo a la dirección postal destino en México, a través de la empresa de mensajería mexicana `Estafeta.com`.

Para el correcto funcionamiento de este Modelo de Negocio es necesaria la integración de funcionalidad que ofrecen `Merkalink.com` y `Estafeta.com`. Dicha integración se implementa mediante servicios Web.

6.3.1 Definición del Proceso Total del Negocio

El primer paso del método es la definición del *Proceso Total del Negocio* (PTN). El PTN es un documento que se establece entre los participantes y que incluye el conjunto de cláusulas que definen la forma en la cual se implementa el Modelo de Negocio.

Por ejemplo, para el caso de estudio mencionado, el PTN incluye las siguientes cláusulas:

1. El cliente deberá autentificarse en `Merkalink.com` y en caso de ser la primera vez que usa la aplicación, deberá registrar sus datos personales incluyendo su dirección postal en México.
2. `Merkalink.com` asignará, automáticamente al cliente, una dirección postal en Laredo, Texas, EEUU. Esta información le será enviada por correo electrónico al cliente.
3. El cliente podrá entonces comprar en alguna tienda en-línea de los EEUU o Canadá y solicitará su envío a la dirección postal en Laredo asignada por `Merkalink.com`.
4. Una vez recibido el artículo en Laredo, el Administrador de `Merkalink.com` registrará el evento en el sistema y le asignará un número de guía que le proporcionará `Estafeta.com`. `Merkalink.com` solicitará a `Estafeta.com` el envío del artículo a México y notificará por correo electrónico al cliente, el número de guía asignado a su paquete para rastreo.
5. En cualquier momento el usuario podrá verificar el estado actual de su envío a través de `Merkalink.com`. Cuando el artículo cruza la frontera, `Estafeta.com` lo notifica automáticamente a `Merkalink.com` y envía a su vez al usuario un mensaje de correo electrónico donde le informa de dicho evento. Una vez cruzada la frontera, el usuario podrá rastrear el estado del envío a través de `Merkalink.com`[21], quien solicitará a `Estafeta.com` el itinerario seguido por el paquete en México.
6. Dentro del período de recepción del artículo, `Merkalink.com` solicitará al cliente, mediante correo electrónico, una evaluación del servicio. Dicha

[21] Esta consulta la podría realizar el usuario también por `Estafeta.com`, pero para efectos del tema explicado en este apartado se hará sólo referencia a `Merkalink.com`.

evaluación se realiza a través de la misma aplicación Web. La evaluación comprende aspectos del servicio, tanto de su empresa como de `Estafeta.com`. Una vez realizada la evaluación por el cliente, `Merkalink.com` envía su resultado a `Estafeta.com`.

6.3.2 Identificación de Procesos Elementales de Negocio y Casos de Uso

El siguiente paso es identificar los *Procesos Elementales de Negocio* (PEN) [51], a partir del PTN, para obtener los *Casos de Uso* (CU) [52]. En su definición original los CU parten del principio de que los Actores tendrán un conjunto de metas[22] que buscarán alcanzar mediante la asistencia del sistema[85]. Dichas metas darán origen a los CU. La aplicación de este mismo principio se ha considerado adecuada también para el caso de las aplicaciones Web, considerando que (1) la teoría de CU es ampliamente comprendida y aceptada para el modelado de requisitos de cualquier tipo de aplicación (2) facilita a los usuarios la validación de los mismos (3) permite corresponder estas metas con la interacción que el usuario tendrá sobre la aplicación Web y (4) dicha interacción será la base para la construcción de los Mapas Navegacionales.

En este contexto, la satisfacción de los requisitos especificados en el PTN llevará a los Actores Humanos a buscar alcanzar sus metas mediante la interacción con la aplicación Web. Las metas podrían estar a diferente nivel de detalle. Sin embargo el nivel que interesa para la futura construcción de la aplicación Web es el que corresponde al concepto de *Tarea del usuario*[23] (de ingeniería de la usabilidad) [51] o *Meta de usuario*[24] en términos de CU [85]. Es el nivel que interesa en los PEN. Un PEN se define entonces como:

> "*Una tarea realizada por una persona en un mismo lugar, en un periodo de tiempo determinado, en respuesta a un evento de negocio, el cual agrega valor medible al negocio y deja los datos en un estado consistente*".

[22] En inglés *goals*
[23] En inglés *user task*
[24] En inglés *user goal*

Cada PEN se corresponderá con un CU, estereotipado con la siguiente sintaxis:

<<UC-*Aplicación*>>

donde *Aplicación* corresponderá al nombre de la aplicación (Merkalink.com o Estafeta.com para el caso de estudio) que ofrece la funcionalidad indicada.

Los *Actores* de cada CU serán de dos tipos: *Humano* y *Aplicación*. Los primeros, corresponderán a los usuarios que interactúan con la aplicación vía la interfaz Web (p.ej. Cliente, Administrador de Merkalink.com y Administrador de Estafeta.com). Los segundos, corresponderán a las aplicaciones que interactúan entre sí (p. ej. Merkalink.com, Estafeta.com o la Tienda en-línea).

Para el caso de estudio, algunos de los PEN son los siguientes:

- **Registro de cliente**: que permite al cliente su autentificación y/o registro de datos personales en Merkalink.com.
- **Compra de artículo**: que representa la compra de un artículo por un Cliente en alguna Tienda en-línea en EEUU o Canadá.
- **Registro de artículo para envío**: solicitud que hace el Administrador de Merkalink.com. Merkalink.com solicita a su vez a Estafeta.com un número de guía para el rastreo del paquete.
- **Consulta de envío**: consulta realizada por el cliente en Merkalink.com sobre el estado actual del envío de algún paquete (rastreo de la ruta seguida por el mismo).
- **Notificación de cruce de frontera**: en la que el Administrador de Estafeta.com captura este evento en Estafeta.com y éste lo notifica automáticamente a Merkalink.com. Además, Estafeta.com envía un mensaje de correo electrónico al Cliente, para indicarle que el artículo ha cruzado la frontera EEUU-México y está listo para su envío a la dirección postal en México.

- **Evaluación del servicio**: donde el `Cliente` evalúa en `Merkalink.com` el servicio que le han brindado las dos compañías. `Merkalink.com` envía su evaluación a `Estafeta.com`.

Toda la funcionalidad implicada en el PTN se esquematiza en un *Diagrama de Casos de Uso* (DCU), en el cual cada PEN se corresponde a un CU. En este diagrama se sigue el convencionalismo de que los *Actores* (*Humanos* o *Aplicación*), que utilizan la funcionalidad de las aplicaciones participantes, se colocan del lado izquierdo y los *Actores aplicación*, cuya funcionalidad es utilizada por las aplicaciones, se colocan del lado derecho (Figura 6.6).

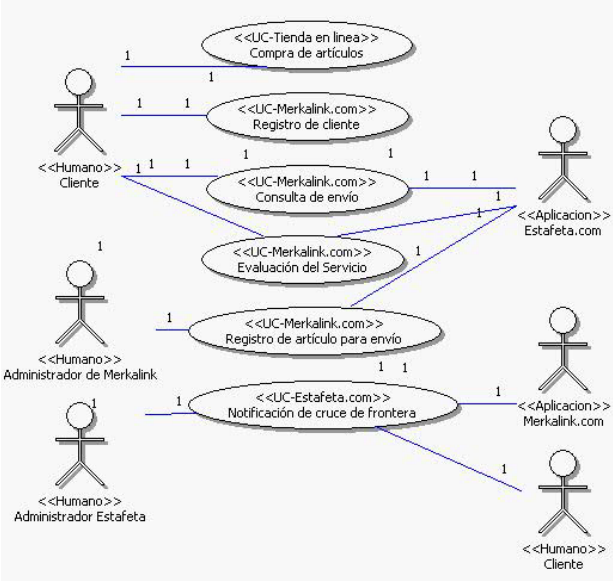


Figura 6.6 Diagrama de Casos de Uso para el Proceso Total del Negocio

6.3.3 Construcción del Diagrama de Actividad Abstracto

Una vez identificados los CU su realización se especifica mediante un *Diagrama de Actividad Abstracto* (DAA). Un DAA es un Diagrama de Actividad UML 2.1 en el cual cada una de sus acciones se define en lenguaje natural en términos de un *Actor* y una *Acción* que realiza el mismo. Las *Acciones* deberán ser alguna de las siguientes:

- **De entrada:** que representa una acción de captura de datos, realizada por el usuario a través de la interfaz Web. Una vez capturados los datos, el usuario invocará a funcionalidad propia o ajena, que los utilizará como parámetros. Se especifica mediante el estereotipo `<<input>>`.
- **De salida:** que representa una acción de salida de datos, los cuales pueden obtenerse a partir de la sociedad de objetos subyacente o a partir de funcionalidad propia o ajena a la aplicación. Se especifica mediante el estereotipo `<<output>>`.
- **De invocación de servicio:** que representa invocación de funcionalidad propia o ajena, resultado de algún paso de entrada. Se especifica mediante el estereotipo `<<service>>`.

Partiendo del DCU se seleccionan los CU de la aplicación Web modelando. Por ejemplo, para el caso de estudio actual, los CU serían los etiquetados con el estereotipo `<<UC-Merkalink.com>>`: *Registro de cliente*, *Consulta de envío*, *Evaluación del servicio* y *Registro de artículo para envío*.

La Figura 6.7 muestra el DAA para el CU *Registro de artículo para envío*, incluyendo las acciones necesarias para su realización: una entrada de datos por la interfaz Web de `Merkalink.com`, la invocación desde `Merkalink.com` a una operación del servicio Web de `Estafeta.com` y el envío de un mensaje de correo electrónico al `Cliente`.

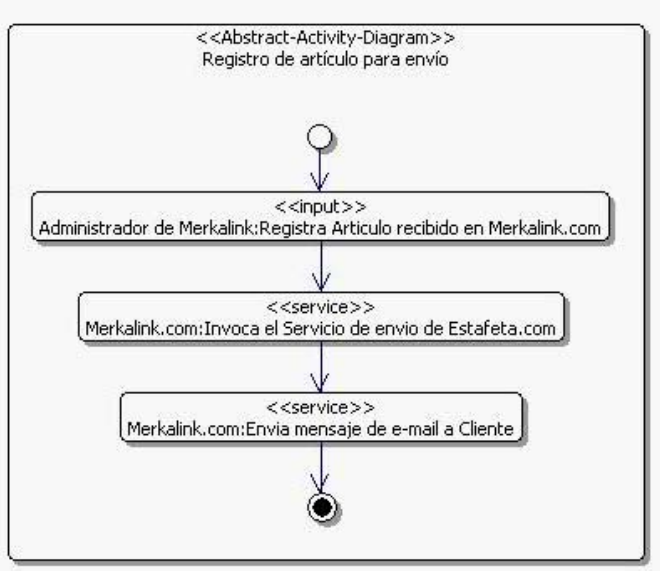


Figura 6.7 DAA para el PEN *Registro de artículo para envío*

6.3.4 Obtención del Diagrama de Clases y el Modelo de Servicios

A partir del conjunto de DAAs el diseñador debe definir los conceptos, operaciones y relaciones que se incluirán como clases, operaciones y relaciones en el *Diagrama de clases* (DC). Además, debe definir los servicios necesarios para lograr la integración entre las aplicaciones e incluirlos en el *Modelo de Servicios* (MS).

La Figura 6.8 muestra el DC para `Merkalink.com` y la Figura 6.9 muestra su MS. En este último se incluye solamente un servicio propio para `Merkalink.com` y un servicio ajeno importado desde `Estafeta.com`. En esta etapa del método, el modelo solamente incluye en cada servicio propio operaciones que representan vistas de las operaciones del DC. Por ejemplo, algunas de las operaciones (como `Entrega.actualizarEstadoActual`)de `Merkalink.com` se incluyen en el MS como vistas (`Serv-Merkalink.actualizarEstadoActualEntrega`).

Es posible que el MS requiera un refinamiento adicional agregando nuevas operaciones compuestas. Para su definición, en las siguientes secciones se establece la regla que permite su identificación y construcción.

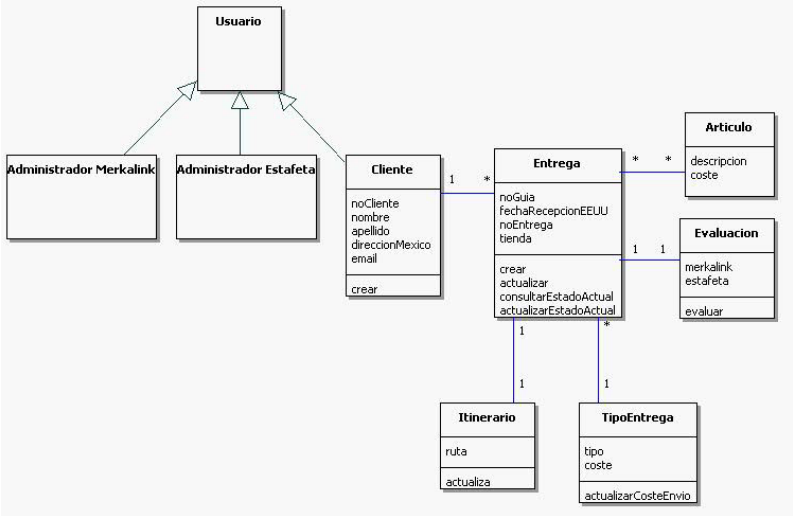


Figura 6.8 El Diagrama de Clases para Merkalink.com

Figura 6.9 El Modelo de Servicios para `Merkalink.com`

6.3.5 Refinamiento del DAA en el Diagrama de Actividad Concreto

Con los modelos creados hasta el momento en el siguiente paso del método el diseñador refina el DAA en un *Diagrama de Actividad Concreto* (DAC), correspondiendo cada acción del primero con una o más acciones del segundo.

Un DAC es un Diagrama de Actividad UML 2.1, en el cual cada una de sus acciones se especifica con una expresión OCL definida sobre los elementos del modelado del DC o el MS. Considerando que la interacción que el usuario tiene con la aplicación conlleva además de entrada y salida de datos también la invocación de servicios, entonces, cada acción del DAC se clasifica como alguna de las siguientes:

- **Acción Web de Entrada:** (AWE) una acción que representa entrada de datos. Estos datos serán parámetros para la invocación de una

operación de clase o de servicio. Se identifica con el estereotipo `<<aw-input>>`.

- **Acción Web de Salida:** (AWS) una acción que representa salida de datos, los cuales pueden ser valores definidos sobre los atributos de objetos o resultados devueltos por la invocación de una operación de un servicio. Se identifica con el estereotipo `<<aw-output>>`.
- **Invocación de Servicio:** (AIS) una acción que representa la invocación de una operación de un servicio. Se identifica con el estereotipo `<<is>>`.

La Figura 6.10 muestra el DAC para el CU *Registro de artículo para envío*. Las correspondencias entre las acciones del DAA y el DAC son las siguientes:

1. La acción *Registra artículo recibido* en `Merkalink.com` se corresponde con la acción AWE `Entrega.crear`, para crear un nuevo objeto `Entrega`.
2. La acción *Invoca el servicio de envío* en `Estafeta.com` se corresponde con una AIS para la ejecución de la operación de servicio ajeno `Serv-Estafeta.enviarArticulo`; ésta, a su vez, devuelve a `Merkalink.com` el número de guía del paquete.
3. Finalmente, la acción *Envío mensaje de e-mail a Cliente* se corresponde con la acción AIS que invoca la operación de servicio propio `Serv-Merkalink.enviarMail` con la cual `Merkalink.com` notifica al cliente el envío del paquete, proporcionándole, vía correo electrónico, el número de guía recibido en la acción anterior.

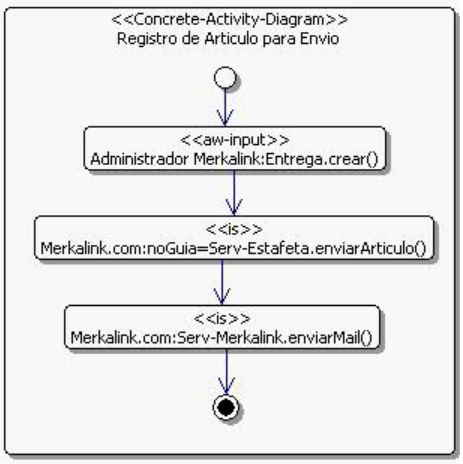


Figura 6.10 El DAC para *Registro de artículo para envío*

6.3.6 Descriptor de Acción Web

Cada una de las Acciones Web del DAC se enriquecen mediante un conjunto de propiedades en un *Descriptor de Acción Web*. Las propiedades del descriptor dependen del tipo de Acción Web (AWE, AWS o AIS) así como también el tipo de entidad a la que se le aplica (clase o servicio). Este descriptor es necesario para definir de manera precisa la Acción Web del DAC y posibilitar con ello la generación del Mapa Navegacional.

Cada descriptor puede ser implementado mediante valores etiquetados[25] de un perfil UML[26] para el DAC. A continuación se introducen.

6.3.6.1 Descriptor para Acción Web en clase

Las *Acciones Web en clases* invocan una operación (acción AWE) sobre algún objeto de la población de una clase del DC, así como también despliegan datos (acción AWS) a partir de los atributos de los objetos de una clase del DC.

En el primer caso, las propiedades para la acción AWE son las siguientes:

1. **ID**: identificador de la AWE. Se utiliza para hacer referencia en descriptores posteriores a las propiedades del descriptor actual.

[25] Una pareja *nombre-valor* que define una propiedad para un estereotipo UML. En inglés *Tag-values*.
[26] Los estereotipos para las diferentes Acciones Web se definen mediante un perfil UML.

2. **Tipo**: con valor *Clase*.
3. **Clase**: nombre de la clase para la cual se aplica el descriptor.
4. **Actor**: actor humano que invoca la operación.
5. **Operación**: operación invocada de la clase.
6. **Resultado**: (opcional) incluye el nombre de la variable que recibe el resultado. En su lugar se puede utilizar también la palabra reservada *Result* (referenciada con la sintaxis <`ID>.Result`).
7. **Parámetros**: lista de parámetros de la operación.
8. **Condición de filtro**: (opcional) condición sobre la población de la clase para seleccionar los objetos candidatos sobre uno de los cuales se desea ejecutar la operación. Se define mediante una expresión OCL.
9. **Patrón de presentación**: (opcional) determina la forma en la que serán presentados los parámetros de la operación. Puede tomar los valores: *Registro* o *Tabular*.

Un ejemplo de este tipo de descriptor para la primer acción AWE del DAC *Registro de Artículo para envío* (Figura 6.10) podría incluir los valores siguientes:

1. **ID**= `AWEC01`.
2. **Tipo**= `Clase`
3. **Clase**=`Entrega`.
4. **Actor**=`Administrador Merkalink`.
5. **Operación**=`crear`.
6. **Paráme-tros**=`noGuia;fechaRecepcionEEUU;noEntrega;tienda`
7. **Patrón de presentación**=*Registro***.**

En el cual se describe que en la implementación de esta Acción Web se solicitarán los parámetros de la operación crear (`noGuia`, `fechaRecepcionEEUU`, `noEntrega` y `Tienda`) en una presentación de *Registro*.

En el segundo caso, las propiedades para la acción AWS son las siguientes:

1. **ID**: un identificador de la AWS. Se puede utilizar para hacer referencia en descriptores posteriores a las propiedades del descriptor actual.

2. **Tipo**: con valor *Clase*.
3. **Clase**: nombre de la clase del DC para la cual se aplica el descriptor.
4. **Atributos**: colección de atributos de la clase. Puede incluir también atributos derivados para mostrar información de clases estructuralmente relacionadas con la actual. Su definición es mediante una expresión OCL.
5. **Condición de filtro**: (opcional) condición sobre la población de la clase para seleccionar los objetos a mostrar. El valor por defecto es toda la población.
6. **Patrón de presentación**: (opcional) determina la forma en la que serán presentados los datos. Puede tomar los valores: *Registro*, *Tabular*, *Maestro-detalle*.

Un ejemplo de este tipo de descriptor está dado para una acción AWS que muestra los datos en presentación *Tabular* de un paquete, cuyo número de guía ha sido capturado en una acción AWE anterior:

1. **ID**= `AWSC01`.
2. **Tipo**= `Clase`
3. **Clase**=`Entrega`.
4. **Atributos**= `noGuia`, `fechaRecepcionEEUU`, `noEntrega`, `self.Itenerario.ruta` (`/RastreoEnvio`).
5. **Condición de filtro**=`Entrega.noGuia = AWEC02.noGuia`.
6. **Patrón de presentación**=*Tabular***.**

6.3.6.2 Descriptor para Acción Web en servicio

Las Acciones Web definidas sobre servicios Web también pueden ser especificadas mediante un descriptor para la invocación de una operación de servicio Web (acción AWE) o para el despliegue de datos provenientes de un servicio Web (AWS).

En el primer caso las propiedades son las siguientes:

1. **ID**: identificador de la AWE.
2. **Tipo**: con el valor *Servicio*.
3. **Servicio**: nombre del servicio propio o ajeno.
4. **Operación**: operación del servicio a invocar.

5. **Resultado**: (opcional) Se puede inicializar con una variable que reciba el resultado (en el caso de una operación *Request-response*) o bien utilizar la palabra reservada *Result* referenciada con la sintaxis <`ID>.Result`).
6. **Parámetros**: lista de parámetros de la operación.
7. **Patrón de presentación**: (opcional) para el caso de operaciones *Request-response*, este patrón permite definir la forma en que se mostrará la respuesta.

Por ejemplo, una acción AWE de servicio para la aplicación `Estafeta.com` que permita solicitar el rastreo de un cierto paquete con un cierto número de guía tendría el descriptor siguiente:

1. **ID**= `AWEC03`.
2. **Tipo**= `Servicio`
3. **Servicio**=`Serv-Estafeta`.
4. **Operación**=`getRastro`.
5. **Parámetros**=noGuia.
6. **Patrón de presentación**=*Registro***.**

En el segundo caso, las propiedades son las siguientes:

1. **ID**: identificador de la AWS.
2. **Tipo**: con el valor *Servicio*.
3. **Servicio**: nombre del servicio propio o ajeno.
4. **Atributos**: colección de datos que se obtienen a partir de operaciones de servicios. Su definición es mediante una expresión OCL.
5. **Patrón de presentación:** determina la forma en la que serán presentados los datos. Puede tomar los valores: *Registro*, *Renglón*.

6.3.7 Ajustes al Modelo de Servicios y al Diagrama de Actividad Concreto

Para la obtención del Mapa Navegacional es necesario que el DAC se exprese solamente en términos de la interacción directa del usuario sobre la interfaz de la aplicación Web (es decir solamente en términos de Acciones Web de Entrada o Salida de la aplicación), por lo que la interacción que la aplicación tendrá con los

Actores Aplicación deberá incluirse como parte de Acciones Web de Entrada para servicios. Para lograr esto se requerirá que el MS y el DAC requieran ajustes adicionales. Estos ajustes incluirán nuevas operaciones compuestas en el MS y una reducción de acciones en el DAC.

La idea básica es: identificar en el DAC secuencias de una o más acciones consecutivas de Acciones Web de Entrada e Invocación de Servicios y reducirlas a una Acción Web de Entrada para la invocación de una operación compuesta de servicio propio. La presencia del siguiente patrón (descrito como expresión regular) de acciones consecutivas en el DAC, indicará la necesidad de la reducción:

awe(is)$^{+}$

Donde:

awe: es una AWE

is: es una AIS

El patrón se describe como: *el conjunto de secuencias de acciones que inician con una Acción Web de Entrada seguida por una o más acciones de Invocación de servicio*. Una vez identificada esta secuencia de acciones, se sustituirá por la acción:

awe-t

Donde:

awe-t: es una AWE para una operación compuesta de servicio propio

La operación compuesta de esta nueva AWE se construye a partir de las operaciones de las acciones originales de Invocación de servicio (**is**), mediante un MDCS. Además, esta operación se agrega al MS en el servicio propio de la aplicación, estableciendo una relación de agregación estática (con propiedad `CTS=Static`) con los servicios participantes restantes, como servicios componente. La acción **awe-t** deberá, además, solicitar al usuario todos los parámetros necesarios para la ejecución de la operación original de la Acción Web de Entrada (**awe**) y la Invocación original de operaciones de servicios (**is**).

La Figura 6.10 es un ejemplo del patrón mencionado. Las tres operaciones del DAC son compuestas en la nueva operación `Serv-Merkalink.registarArtparaEnvio()`, sustituyendo las tres acciones originales (Figura 6.11).

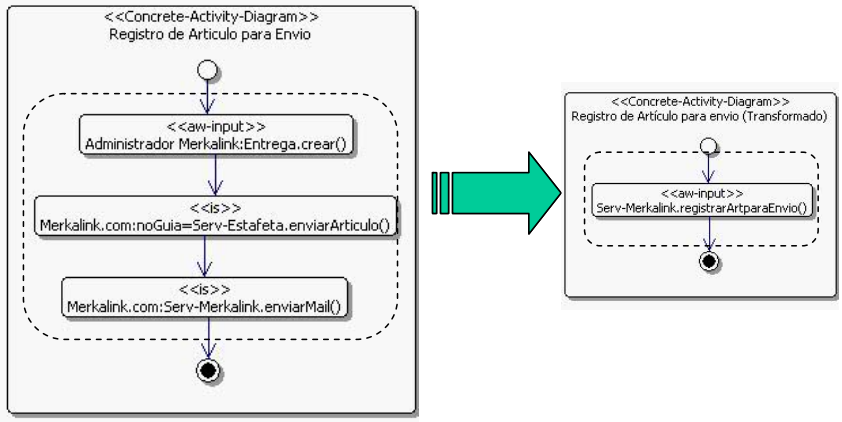


Figura 6.11 Transformación del DAC

La nueva operación compuesta se incluye como una operación más del MS, se agrega como servicio componente a `Serv-Estafeta` y se define su lógica mediante el MDCS de la Figura 6.12.

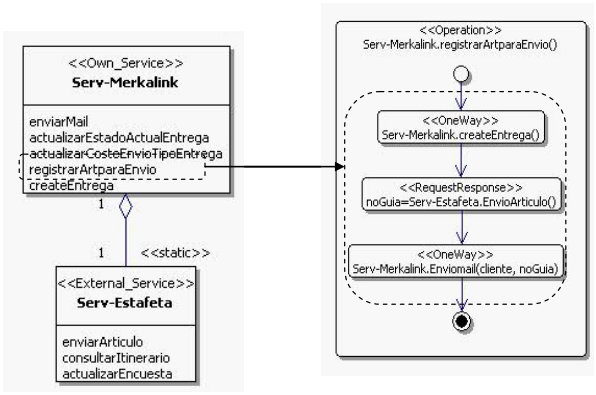


Figura 6.12 El MS y el MDCS para la nueva operación compuesta

6.3.8 Obtención del Mapa Navegacional

A partir de los modelos obtenidos hasta este paso del método, es posible obtener un Mapa Navegacional prototipo para la aplicación Web. Este Mapa Navegacional podrá ser revisado y modificado (si es necesario) posteriormente por el diseñador, para luego transformarse en la aplicación Web final.

A continuación se ofrece un conjunto de reglas, para las dos fases principales del método OOWS, que permiten la detección de cada una de las primitivas del Mapa Navegacional.

6.3.8.1 Fase Autoreo-a-escala-mayor

Cada uno de los Actores Humanos, el Diagrama de Usuarios y el Mapa Navegacional son obtenidos de la siguiente manera:

1. **Diagrama de Usuarios:** se identifican en el *Diagrama de Casos de Uso* los Actores Humanos de la aplicación y se clasifican como `Anónimo`, `Genérico` o `Registrado`. Una vez hecha esta clasificación se incluyen en el *Diagrama de Usuarios* (DU). Por ejemplo, la Figura 6.13 muestra el DU para `Merkalink.com`.

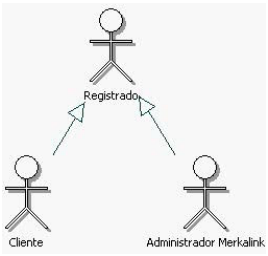


Figura 6.13 Diagrama de Usuarios para `Merkalink.com`

2. **Mapa Navegacional**: considerando los DAC de cada uno de los CU del Actor Humano, se construyen cada una de las secciones del Mapa Navegacional, a partir de las siguientes reglas de:

   a. **Detección de contextos navegacionales**:

      1. Por cada AWE (de clase o servicio), se define un contexto navegacional nombrado con el valor de la propiedad *Operación* del descriptor de la Acción Web.
      2. Por cada AWS de clase, se define un contexto navegacional nombrado con el valor de la propiedad *Clase* del descriptor de la Acción Web.

3. Por cada AWS de servicio, se define un contexto navegacional nombrado con el valor combinado de la propiedad *Servicio* y alguno de sus atributos. Esta regla podría requerir la intervención manual del diseñador para seleccionar el atributo más adecuado o proporcionar un nombre por defecto.
4. El contexto navegacional que se obtiene a partir de la primer Acción Web se etiqueta como de *Exploración* y se nombra con el identificador del DAC[27]. Los contextos navegacionales restantes se etiquetan como de *Secuencia*.
5. Defínase como contexto navegacional *Home* candidato, a aquél que se generó a partir del primer DAC del PTN.

b. **Detección de relaciones navegacionales**:

1. Defina una relación navegacional entre un par de contextos navegacionales, para cada par de Acciones Web consecutivas.

Las Figuras 6.14 y 6.15 muestran el Mapa Navegacional para el actor `Cliente` y su reificación como página Web. Incluye tres contextos de *Exploración*: `Registro de cliente`, `Consulta de envío` y `Evaluación del Servicio`, correspondientes a los casos de uso del Actor Humano `Cliente`. El DAC de cada caso de uso se ha utilizado para obtener una sección del Mapa.

Las Acciones Web para la realización del CU *Registro de cliente* son las primeras en la especificación del contrato, por lo que su contexto es etiquetado como `Home`. A partir del caso de uso *Consulta de envío* se define el contexto de *Exploración* `Consulta de envío` obtenido por la AWE `Entrega.consultarEstadoActual()`. El contexto *Secuencial* `Itinerario` se obtiene a partir de la AWS que despliega los datos de la ruta del paquete (`Itinerario.ruta`).

[27] Se renombra si por las reglas anteriores ya había sido nombrado o si el identificador no es adecuado se requerirá la intervención manual del diseñador.

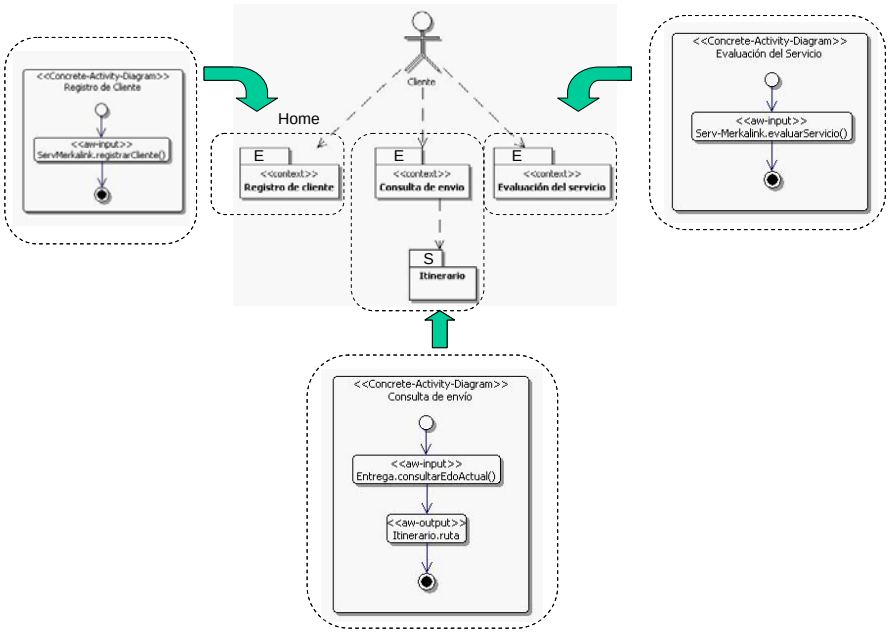


Figura 6.14 Mapa Navegacional para el Actor `Cliente`

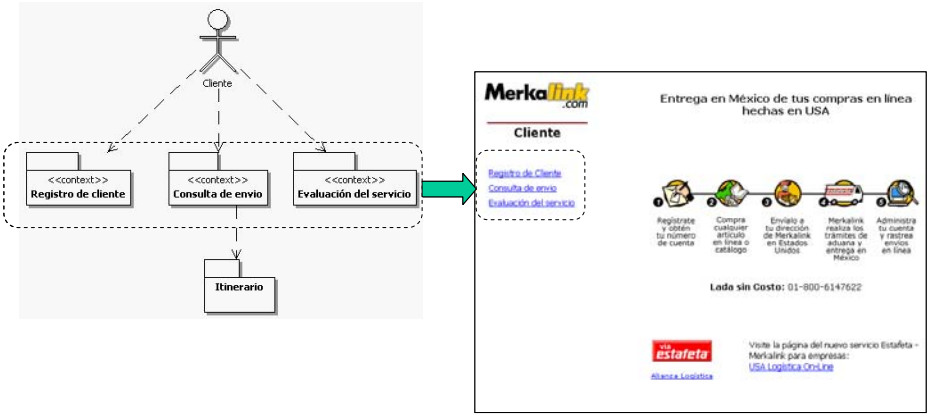


Figura 6.15 Reificación del Mapa Navegacional como página Web

6.3.8.2 Fase Autoreo-a-escala-menor

Una vez concluida la fase anterior, las primitivas que se incluyen en el detalle de cada uno de los contextos navegacionales del Mapa Navegacional se detectan a partir de las Acciones Web del DAC mediante las siguientes reglas de:

1. **Detección de Unidades de Información**:
    a. **IU Directora**: se incluye en cada contexto a partir de una acción AWS en clase, nombrándola con el valor de la propiedad *Clase* del descriptor de la Acción Web. Los atributos que incluye son todos los definidos en la propiedad *Atributos* del mismo descriptor, excepto los atributos derivados. A partir de la propiedad *Condición de filtro* se establece la condición de población en la IU para seleccionar los objetos a mostrar.
    b. **IU Complementaria**: se incluye en cada contexto obtenido a partir de una acción AWS en clase, considerando los atributos derivados definidos en la propiedad *Atributos* de su descriptor.
    c. **Operación**: se incluye en cada contexto obtenido a partir de una acción AWE en clase. La IU que la contendrá se nombra con el valor de la propiedad *Clase* del descriptor. La operación se incluye en la IU nombrándola con el valor de la propiedad *Operación*, incluyendo la lista de parámetros de la propiedad *Parámetros*. A partir de la propiedad *Condición de filtro* se establece la condición de población en la IU, de los objetos candidatos a uno de los cuales se le aplicará la operación.

2. **Detección de Unidades de Funcionalidad:** se incluyen en los contextos navegacionales que se han obtenido a partir de alguna acción Web (AWE o AWS) en servicio. Los elementos de la FU se definen de la siguiente manera:
    a. **Atributos**: se incluyen en la FU de un contexto a partir de una acción AWS en servicio. La FU se nombra con el valor de la propiedad *Servicio* del descriptor. Los atributos que incluirá la FU son todos los definidos en la propiedad *Atributos* del mismo descriptor.
    b. **Operación**: se incluye en la FU de un contexto obtenido a partir de una acción AWE en servicio. La FU se nombra con el valor de la propiedad *Servicio* del descriptor. La operación se incluye en la FU nombrándola con el valor de la propiedad *Operación*, incluyendo la lista de parámetros de la propiedad *Parámetros*.

3. **Detección de relaciones navegacionales**:
    a. **De dependencia contextual:** se establecen entre una IU directora y una IU complementaria, de un contexto navegacional obtenido a partir de una ac-

ción AWS que incluye un atributo derivado en la propiedad *Atributos* de su descriptor (regla 1.a).

b. **De contexto**: se establece entre un par de IU, obtenidas a partir de clases estructuralmente relacionadas y que se han obtenido a partir de un par de acciones AWE y AWS consecutivas. El contexto destino de la relación será aquél que se ha definido a partir de la AWS.

c. **Enlace de servicio:** se defíne para el caso de un par de acciones AWE consecutivas, que se han correspondido con un par de contextos navegacionales igualmente consecutivos (de acuerdo a la línea guía 2.a.1 de la Fase *Autoreo-a-escala-mayor*). Inclúyase un *Enlace de servicio* en la operación del primer contexto, que apunte al segundo contexto navegacional.

La Figura 6.16 muestra el contexto navegacional `Registro de cliente` definido a partir de la AWE `Serv-Merkalink.registrarCliente()`. Aplicando la regla 2.b se incluye en el contexto navegacional una FU con la operación mencionada la cual crea un nuevo objeto `Cliente` y además envía un mensaje de correo electrónico al cliente registrado donde se le informa su dirección postal en Laredo, Texas.

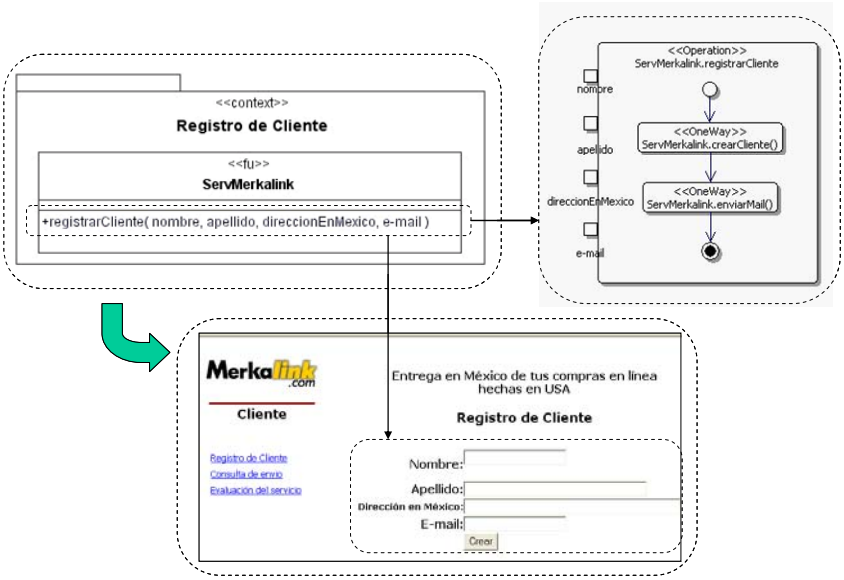


Figura 6.16 Contexto navegacional para el caso de uso *Registro de cliente*

La Figura 6.17 muestra la aplicación de la regla 3.b para el DAC `Consulta de envíos`. La realización de su CU se implementa seleccionando primero el paquete (o `numEntrega`) y desplegando posteriormente el itinerario seguido hasta el momento por el mismo.

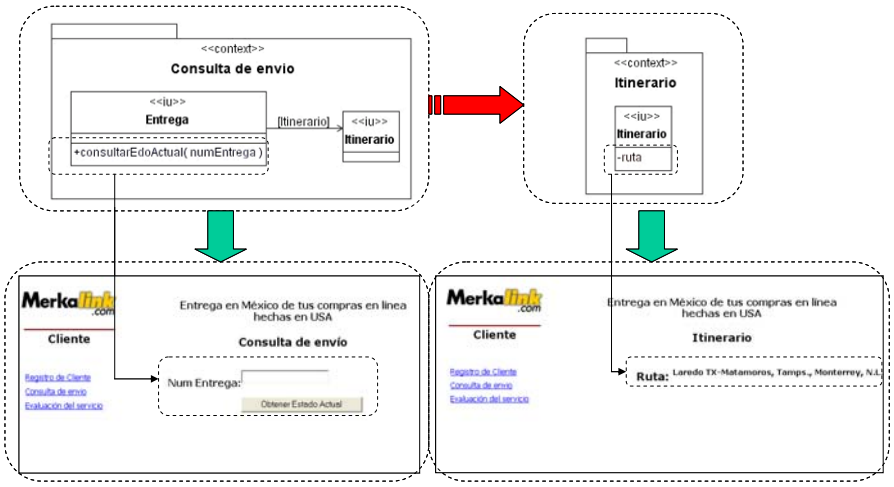


Figura 6.17 Contextos navegacionales para el caso de uso Consulta de envíos

6.3.9 Modelo de Presentación

Las propiedades *Patrón de Presentación* de los descriptores de las acciones AWE y AWS en clase, se utilizan para la construcción del *Modelo de Presentación* de la aplicación Web. A cada una de las IU que se han definido en el Modelo de Navegación se les aplica el patrón definido en el descriptor de la acción Web que le dio origen. La misma propiedad se utiliza y aplica para las FU obtenidas a partir de acciones AWS en servicios.

6.4 Conclusiones

Se ha presentado un método para la obtención de un Mapa Navegacional OOWS a partir de un modelo de proceso Negocio-a-Negocio en el cual participan actores humanos y aplicaciones externas. Los pasos permiten al ir desde la captura de requisitos hasta la obtención del Mapa Navegacional OOWS que les da soporte. El método sigue una estrategia semi-automática, con algunos puntos en los cuales es necesa-

rio la revisión o ajuste de los artefactos obtenidos, por el analista y el diseñador. La solución que se ofrece es un punto de partida para la obtención de un Mapa Navegacional prototipo. Otras estrategias metodológicas son posibles. Se plantearán en la sección de trabajo futuro de esta tesis.

CAPÍTULO 7
Generación de código Java y BPEL a partir de los Modelos de Servicios y Composición

El siguiente capítulo explica la estrategia de generación automática de código partiendo de los *Modelos de Servicios* y *Dinámico de Composición de Servicios*. La estrategia se basa en el marco de trabajo definido por la Arquitectura Dirigida por Modelos (*Model-driven Architecture-MDA*) y se ha organizado en cuatro escenarios de transformación de modelos a modelos y de modelos a código.

En cada escenario se definen los metamodelos independientes de plataforma (*Platform Independent Models-PIM*), se seleccionan las plataformas tecnológicas para la definición de metamodelos específicos de la plataforma (*Platform Specific Models-PSM*) y se definen las reglas de transformación modelo a modelo y modelo a código.

Esta estrategia de generación de código se suma a la ya existente en el método OOWS, de tal manera que los componentes software generados se acoplan con los generados anteriormente.

El capítulo está organizado en dos partes: en la primera se explica de forma general la estrategia de generación de código, los escenarios y los metamodelos completos PIM y PSM requeridos en todos los escenarios. En la segunda parte se explica de forma detallada cada escenario, incluyendo las secciones de los metamodelos PIM y PSM requeridos en el mismo, así como sus transformaciones particulares modelo a modelo y modelo a código.

7.1 Estrategia general de generación de código, escenarios y metamodelos

La estrategia general para la generación de código en todos los escenarios está organizada en tres pasos de transformación (Figura 7.1): (1) *Modelo-A-Modelo*; (2) *Modelo-A-Texto* y (3) *Texto-A-Binario*.

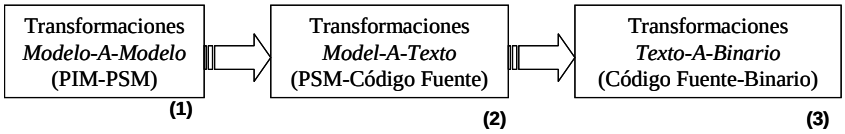


Figura 7.1 Estrategia general de generación de código

En la primera transformación (*Modelo-A-Modelo* – ver Figura 7.2) se parte de la definición de los modelos PIM de *Servicio* y *Dinámico de Composición de Servicios* y se establece un conjunto de transformaciones con modelos PSM para las plataformas Java, Axis[53] y WS-BPEL 1.1 (de aquí en adelante BPEL). Estas transformaciones se establecen utilizando el lenguaje de mapeo operacional del lenguaje QVT operacional de Together [56] (*Query-View-Transformations*) buscando corresponder las abstracciones conceptuales de los modelos PIM con las abstracciones de las plataformas tecnológicas seleccionadas.

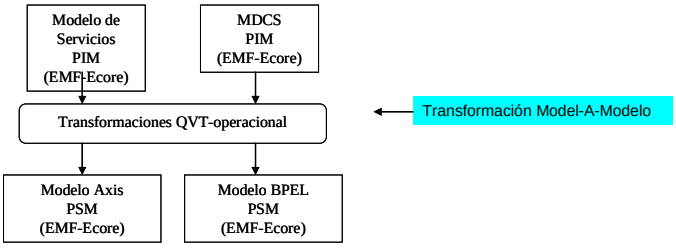


Figura 7.2 Estrategia de transformación *Modelo-A-Modelo*

En el segundo paso se define un conjunto de transformaciones *Modelo-A-Texto* (Figura 7.3), basadas en el uso de plantillas, desde los modelos PSM hacia código Java, Axis y BPEL.

Finalmente, en el tercer paso se utiliza el compilador Java para traducir el código Java y Axis a Java-*bytecodes*. El código BPEL es montado directamente en un motor de ejecución BPEL. Este código no requiere compilación adicional, la realiza en tiempo de ejecución el servidor de aplicaciones.

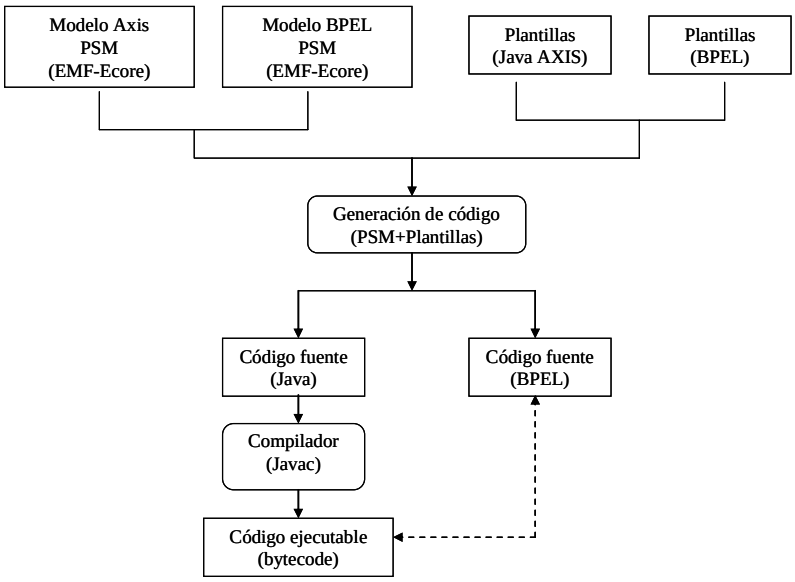


Figura 7.3 Estrategia de transformación Modelo-A-Texto y generación de código

7.1.1 Escenarios de transformación de modelos

El proceso de generación de código se ha dividido en cuatro escenarios de transformación de modelos. Cada escenario se aborda de manera individual, aunque algunos casos dependen de otros y requieren de su aplicación previa para la realización de su estrategia de generación de código.

De manera resumida los escenarios son los siguientes:

1. **Importación de Servicios Ajenos:** el cual considera la importación (a partir de su interfaz WSDL) de servicios ajenos a la aplicación

Web. Este caso considera la generación de código Java que pueda acoplarse mediante *stub's* generados para el *framework* AXIS y a su vez genera una fachada que pueda acoplarse con los componentes software restantes (como aquellos que representen páginas Web – por ej. páginas JSP-).

2. **Generación de Servicios Propios como vistas de Clases de Negocio:** en el cual una primera transformación de modelos obtiene a las operaciones de las clases del diagrama de clases (objetos de negocio) como operaciones de un nuevo servicio propio. Luego, otra transformación de modelos genera las clases Java para la implementación del *skeleton* e interfaz WSDL necesarios para que dicho servicio propio pueda ser consumido por otras aplicaciones. Finalmente, se realiza una última transformación que genera una fachada para permitir el acoplamiento del nuevo servicio propio con los componentes software restantes de la aplicación.
3. **Agregación y Composición de Servicios:** en el cual se considera la composición de servicios a partir de los modelos estructural y dinámico de servicios. Mediante un conjunto de transformaciones de modelos se obtiene un modelo específico para la plataforma BPEL que implementa el servicio compuesto. Posteriormente, partiendo de este modelo específico de plataforma, se genera todo el código necesario para el montaje del servicio compuesto en un motor de ejecución BPEL.
4. **Especialización de Servicios:** el cual implementa la especialización de un servicio propio o ajeno (base) en otro servicio propio (derivado). Una primera transformación de modelos convierte los servicios base y especializado en una agregación de servicios. Después, a esta agregación de servicios se le aplican las transformaciones definidas para el escenario de agregación y composición de servicios anterior, para así obtener todo el código necesario que permita su montaje en el motor de ejecución BPEL.

7.1.2 Metamodelos PIM

Se han definido dos metamodelos PIM: uno para el *Modelo de Servicios* y otro para el *Modelo Dinámico de Composición de Servicios*. Éstos capturan aspectos distintos, sin embargo, por las limitaciones técnicas de las herramientas de transformación de modelos seleccionados (Eclipse Modeling Framework, EMF[55] y QVT operacional de Together[56]) se han tenido que incluir ambos, en la práctica, en un solo metamodelo. Por ejemplo, en la herramienta *Borland Together* cada transformación de modelos QVT operacional sólo permite un solo modelo de entrada, aunque en ocasiones es necesario utilizar dos. Así, la única forma para implementar la transformación es mediante un solo modelo que incluya de manera conjunta los dos modelos.

La Figura 7.4 muestra el metamodelo PIM (EMF) indicando la parte que corresponde al *Modelo de Servicios* y la parte que corresponde al *Modelo Dinámico de Composición de Servicios*. Algunos pocos cambios han tenido que realizarse respecto a la propuesta original de metamodelos de capítulos anteriores. Por ejemplo, la implementación EMF del MDCS ha incluido para cada acción (*Action*), el servicio (*ServiceAc*) cuya operación se invoca o es invocada. Con todo, la esencia de la definición original de los metamodelos se mantiene.

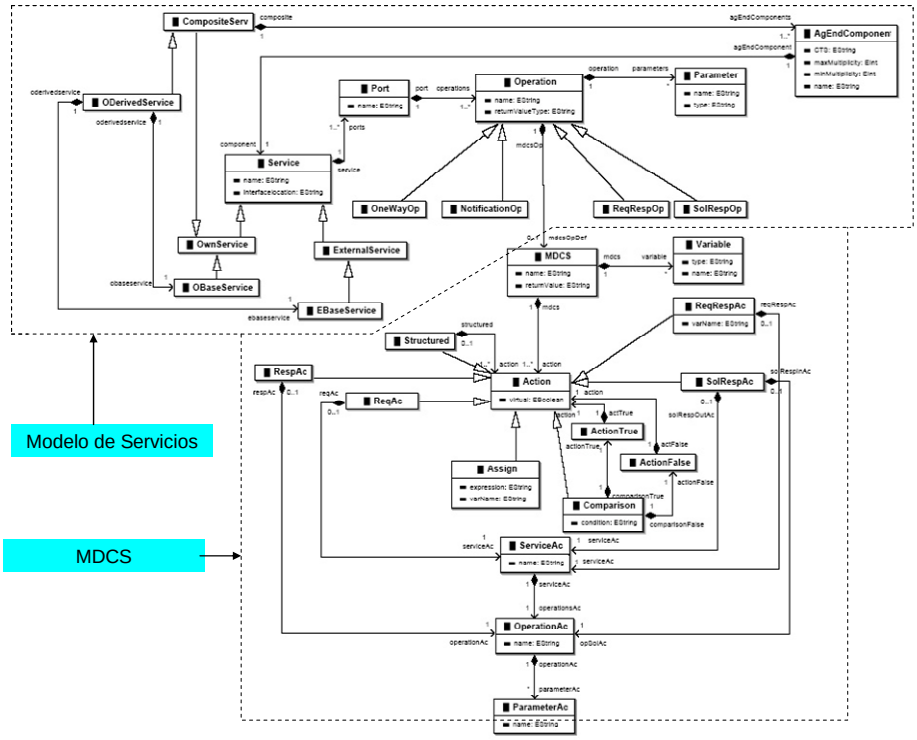


Figura 7.4 El metamodelo PIM de Servicios implementado mediante EMF

En secciones posteriores se irá explicando para cada escenario la forma en la que son utilizadas las diversas partes de este metamodelo.

7.1.3 Metamodelos PSM

Se han seleccionado las plataformas tecnológicas Java 1.4, Axis 1.1 y WS-BPEL 1.1 para la generación de código. Para cada una de ellas se han definido metamodelos PSM que incluyen solamente un subconjunto de las abstracciones necesarias para el modelado de la plataforma.

La Figura 7.5 muestra el metamodelo PSM para Java y Axis. Las abstracciones incluidas permiten la generación de los *stub's* y *skeleton's* necesarios, así como también las clases Java fachada que permitirán el acoplamiento con los componentes software generados con los componentes antes obtenidos en la estrategia original de generación de código de OOWS.

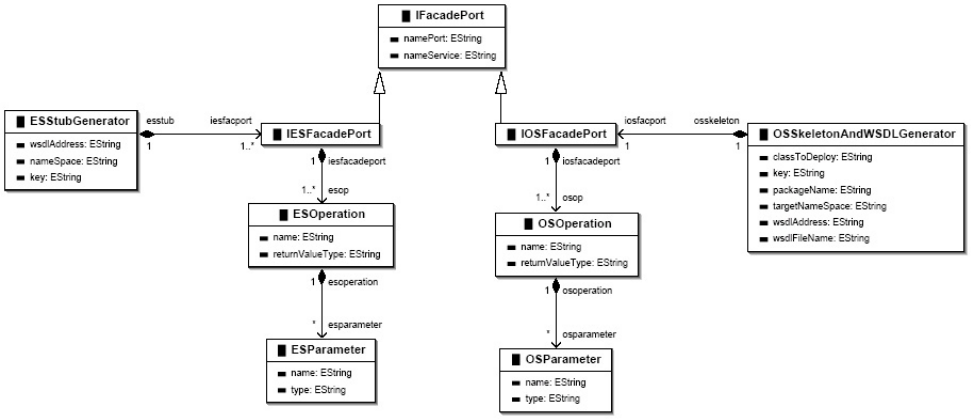


Figura 7.5 Metamodelo PSM para Java y Axis

Las Figuras 7.6 y 7.7 muestran el metamodelo PSM para BPEL. Igualmente representa sólo un subconjunto de toda la plataforma, ya que la implementación tecnológica del MDCS no requiere de todas las primitivas tecnológicas que ofrece BPEL.

Por las limitaciones técnicas mencionadas que ofrece la implementación del lenguaje de transformación seleccionado, QVT operacional de Together, se incluyen abstracciones para la generación de las clases Java fachada junto con las abstracciones para la generación del código BPEL. La Figura 7.6 muestra la parte del metamodelo para la generación de las clases Java fachada y la Figura 7.7 muestra la parte del metamodelo para la generación del código BPEL.

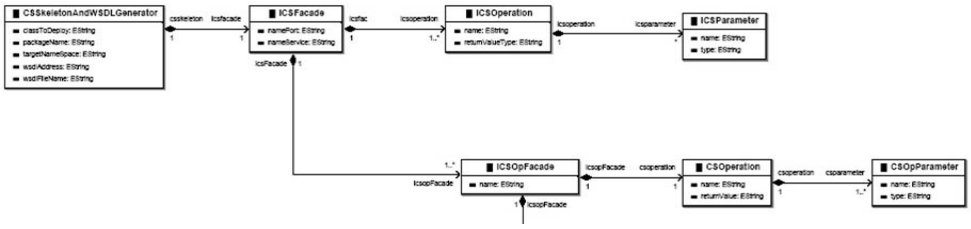


Figura 7.6 Metamodelo PSM-BPEL para la generación de clases fachada

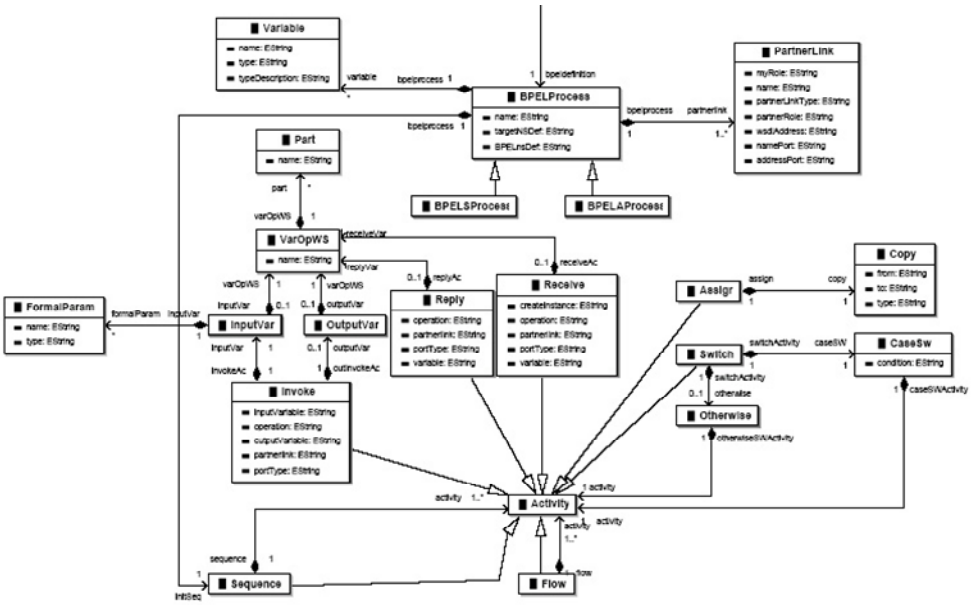


Figura 7.7 Metamodelo PSM-BPEL para la generación de procesos BPEL

Los modelos PIM son transformados hacia estos modelos PSM y estos últimos, mediante un conjunto de transformaciones *Modelo-A-Texto,* a código final que implementa los diversos escenarios. Los detalles de estos procesos de transformación se explicarán en la siguiente sección.

7.2 Implementación de escenarios

En la siguiente sección se detallan cada uno de los escenarios de transformación de modelos y generación de código. Cada escenario se presenta bajo la siguiente estructura: primero se detallan los pasos que conforman la estrategia, después se explican cada uno de los metamodelos PIM y PSM utilizados en el escenario y finalmente se introducen las transformaciones *Modelo-A-Modelo* y *Modelo-A-Texto* para la generación de código final.

7.2.1 Escenario 1: Importación de Servicios Ajenos

La importación de servicios ajenos se realiza en tres pasos:

(1) En el **primer paso** (ver Figura 7.8), se lleva a cabo la importación del servicio ajeno al PIM de Servicios. Este paso se implementa como un proceso de importación *Texto-A-Modelo*, mediante el cual se genera en el Modelo de Servicios un servicio ajeno a partir de los elementos de la interfaz WSDL del servicio.

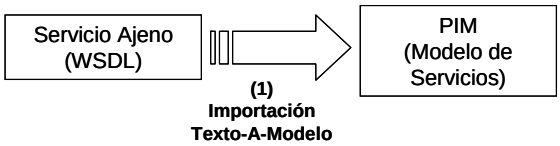


Figura 7.8 Primer paso : importación del servicio ajeno al Modelo de Servicios

(2) En el **segundo paso** (ver Figura 7.9), el PIM de Servicios se transforma en los PSM de Servicios para las plataformas Java y Axis mediante un conjunto de reglas *Modelo-A-Modelo*. Estas transformaciones corresponden el servicio ajeno con abstracciones del metamodelo PSM que posibilitarán en el paso siguiente la generación de clases *stub* (en Java) de acuerdo a las reglas que se especifican en el *framework* Axis. Las transformaciones corresponderán también al servicio ajeno con abstracciones del PSM que posibilitarán posteriormente la generación de clases fachada para el acoplamiento de las clases *stub* con otros componentes software de la aplicación.

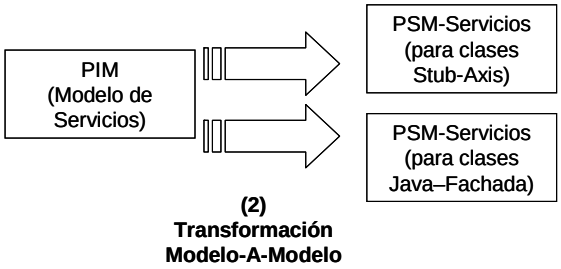


Figura 7.9 Segundo paso: transformación del Modelo de Servicios al PSM de Servicios

(3) En el **tercer paso** (ver Figura 7.10), se realiza la generación de código final. Partiendo de los PSM de Servicios obtenidos en el paso anterior se generan las clases Java para el *stub* y fachada mencionados. Estas clases son generadas median-

te un conjunto de transformaciones *Modelo-A-Texto*. Luego las clases obtenidas son compiladas mediante el compilador Java.

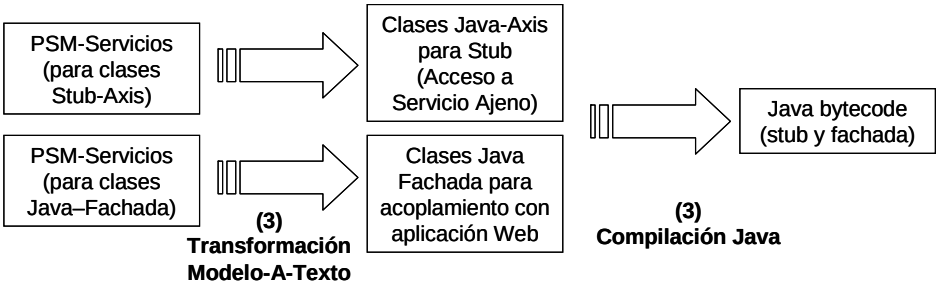


Figura 7.10 Tercer paso: generación de código final

7.2.1.1 Metamodelo PIM

La sección del PIM de Servicios que se utiliza para este escenario se muestra en la Figura 7.11. Las metaclases (que se nombrarán solamente como *clases*) que se utilizan son las siguientes:

- **Service**: una clase padre para todos los tipos de servicios. Posee como atributos el nombre del servicio (`name`) y la dirección de la interfaz del servicio (`interfacelocation`, la dirección URL de su interfaz WSDL).
- **Port**: cada servicio puede tener uno o más puertos. El puerto solamente incluye su nombre (`name`) como atributo.
- **Operation**: cada puerto puede tener una o más operaciones. Cada operación posee como atributos su nombre (`name`) y el tipo del valor de retorno (`returnValueType`). A su vez las operaciones se especializan en *One Way* (`OneWayOp`), *Notification* (`NotificationOp`), *Request-response* (`ReqRespOp`) y *Solicit-response* (`SolRespOp`).
- **Parameter**: cada operación tiene cero o más parámetros, cada uno de los cuales posee dos atributos: su nombre (`name`) y tipo (`type`).

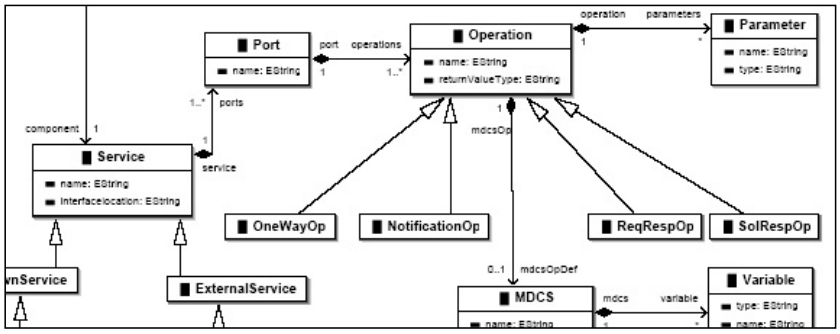


Figura 7.11 Extracto del PIM de Servicios que incluye las metaclases para servicios ajenos

Un ejemplo del PIM de Servicios está dado en la Figura 7.12. Se muestra para el servicio de búsquedas de Google.

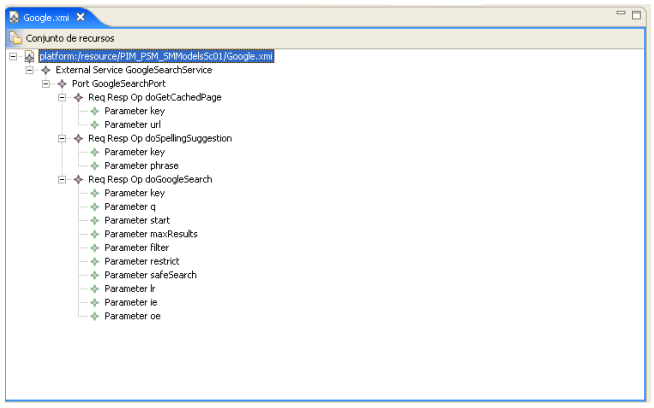


Figura 7.12 Modelo PIM de Servicios para Google

7.2.1.2 Metamodelo PSM

El PIM de Servicios se corresponde con el PSM de Servicios para las plataformas Java y Axis (ver Figura 7.13). Este PSM de Servicios se ha diseñado considerando las metaclases necesarias para la generación de código. Representa un subconjunto suficiente de las plataformas mencionadas para la implementación de servicios ajenos.

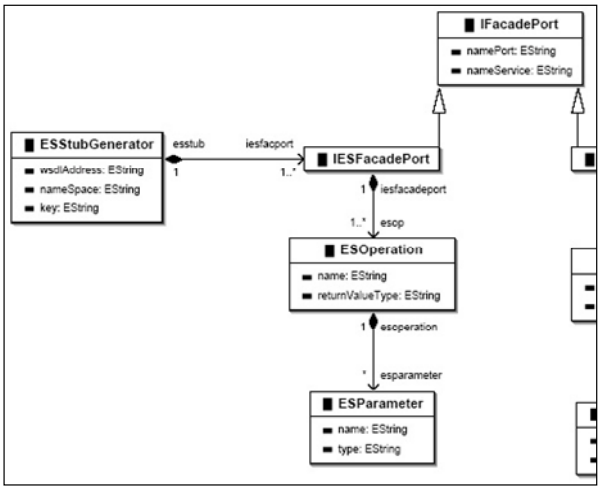


Figura 7.13 Extracto del PSM de Servicios incluyendo las metaclases para implementación de servicios ajenos en Java y Axis

Las clases que se han incluido en el metamodelo son las siguientes:

- **ESStubGenerator**: una clase *fabricación pura*[28] repositorio de valores necesarios para la generación de código Axis. Posee los siguientes atributos: la dirección URL de la interfaz WSDL del servicio Web (`wsdlAddress`); el espacio de nombres (`nameSpace`) para el paquete que contendrá las clases generadas y el atributo llave (`key`), en caso de que el servicio Web requiera de alguna clave especial para su invocación (requisito común de algunos servicios Web, como en `Google` o `Amazon`).
- **IESFacadePort**: por cada puerto del servicio ajeno se generará una fachada con las operaciones del mismo. Esta clase es una especialización de la clase `IFacadePort`, clase padre para los dos tipos de puerto que se han definido (para servicios propios y ajenos). Incluye dos atributos: el nombre del puerto (`namePort`) y el nombre del servicio (`nameService`). Su uso combinado da origen al nombre de la clase fachada.

[28] Un tipo de clase que no tiene correspondencia directa con alguna abstracción de las plataformas tecnológicas modeladas, pero que se ha diseñado por conveniencia.

- **ESOperation**: esta clase se utiliza para la generación de las operaciones de cada fachada. Posee dos atributos: el nombre de la operación (`name`) y el valor de retorno (`returnValueType`).
- **ESParamater**: los parámetros de cada operación de la fachada se definen en esta clase. Posee dos atributos: el nombre del parámetro (`name`) y su tipo (`type`).

La Figura 7.14 muestra un ejmplo del PSM de servicios obtenido a partir del PIM de servicios de Google.

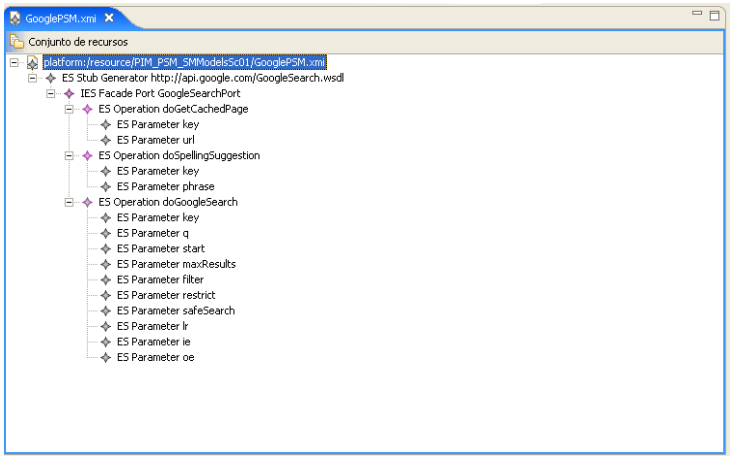


Figura 7.14 Modelo PSM de servicios para Google

7.2.1.3 Reglas de transformación PIM-A-PSM

El conjunto de reglas de transformación del metamodelo PIM de Servicios al metamodelo PSM de Servicios están indicados en la Figura 7.15. En esta figura se señalan, para cada regla de transformación, la metaclase origen y la metaclase destino.

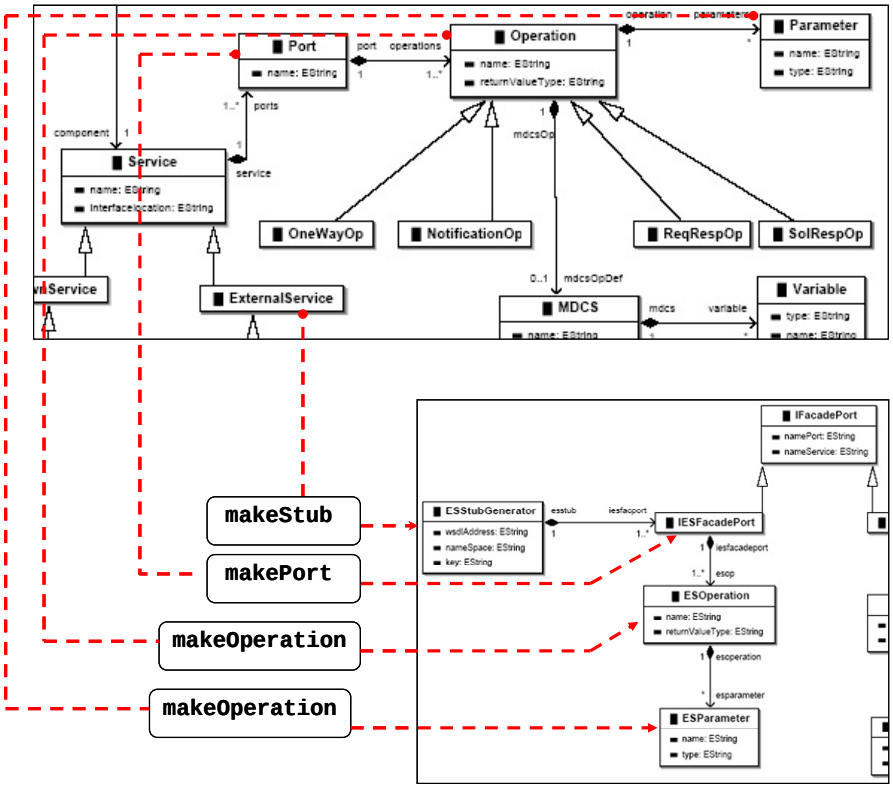


Figura 7.15 Transformaciones del PIM de Servicios al PSM de Servicios

Cada una de las reglas de transformación son las siguientes:

1. **makeStub**: por cada servicio ajeno (`ExternalService`) se crea un generador de *stub* (`ESStubGenerator`), inicializando su dirección WSDL desde la localización de su interfaz (`Service.interfacelocation`). Invoca la regla de transformación *makePort* para la generación de los puertos del servicio.
2. **makePort**: por cada puerto del servicio ajeno (`Port`) se crea una fachada (`IESFacadePort`). El nombre de la fachada esta dado por la concatenación del nombre del servicio (`Port.name`) y el nombre del puerto (`service.name`). Invoca la regla de transformación *makeOperation* para la generación de las operaciones del puerto.
3. **makeOperation**: crea cada una de las operaciones (`ESOperation`) de la fachada, inicializando su nombre (`name`) y creando sus parámetros mediante la invocación de la regla *makeParameter*.

4. **makeParameter**: genera cada uno de los parámetros de la operación (`ESParameter`) a partir del nombre (`Parameter.name`) y tipo (`Parameter.type`) del parámetro del servicio ajeno.

Como un ejemplo de implementación de estas transformaciones, la Figura 7.16 muestra la transformación `makeStub`, programada en QVT-operacional de Together. En esta regla se recibe en su entrada un objeto de la clase *ExternalService* y a partir del mismo se genera el objeto de la clase *ESStubGenerator* inicializando sus atributos `wsdlAddress` y `iesfacport`. Én este último atributo se agregan todos los puertos de la fachada.

La implementación de otras transformaciones se incluye en el Apéndice 1.

```
transformation Kernel_To_Psm_ws;
metamodel 'http://kernel';
metamodel 'http://psm_ws';
/* Del Servicio Externo (PIM) al Stub Generator (PSM) ------------------------------- */
mapping main(in model: kernel::ExternalService): psm_ws::ESStubGenerator {
          init {
                    result := makeStub(model);
          }
          object {
          }
}

mapping makeStub(in model: kernel::ExternalService): psm_ws::ESStubGenerator {
          init {
                    var allPorts := model.ports;
          }
          object {
                    wsdlAddress := model.interfacelocation;
                    iesfacport  += allPorts->collect(p|makePort(p));
          }
}
```

Figura 7.16 Transformación *Modelo-A-Modelo* `makeStub` en QVT-operacional

7.2.1.4 Reglas de transformación PSM-A-Código

Una vez que se han generado las clases del PSM de servicios es posible la obtención del código final. El código generado se organiza así: por cada puerto del servicio ajeno se genera una clase Java fachada que utiliza un conjunto de clases Java-Axis que implementan el *stub* para el consumo del servicio ajeno. La fachada ofrece en su interfaz las operaciones del puerto y utiliza el *stub* para la invocación del servicio remoto.

Un conjunto de reglas de transformación *Modelo-A-Texto* permiten la generación del código Java para cada una de las fachadas. Cada fachada será generada

haciendo uso de las reglas que definen el uso de las clases Java para Axis. A su vez, haciendo uso de los valores almacenados en los atributos de la clase `ESStubGenerator`, la utilería `wsdl2Java` de Axis genera el conjunto de clases *stub* necesarias para cada fachada.

La transformación se define, en lenguaje natural, de la siguiente manera:

1. Por cada puerto del generador de *stub* (`ESStubGenerator`):
    a. Genera en el código de clase Java fachada:
        i. La importación de clases de soporte (manejo de acceso remoto-RMI y excepciones) y los atributos Java.
        ii. El constructor de la clase, dentro del cual se invocará a una operación de soporte (infraestructura) para el acceso al puerto.
        iii. El código para el acceso a cada una de las operaciones del servicio ajeno.
        iv. La operación de soporte para el acceso al puerto.
2. A partir de los valores del generador *stub*, y utilizando la herramienta `wsdl2java`, se generan las clases *stub* para el servicio ajeno.

La Figura 7.17 muestra un extracto de la programación en *MofScript* de esta transformación: a partir de cada puerto de la fachada (`iesfacport`) se genera una clase Java que implementa al mismo. Una vez que se genera la fachada, se invocan a las herramientas de generación de código Java del *framework* Axis.

La programación completa de la transformación se incluye en el Apéndice 1.

```
import "IESFacadePort2JavaSc01.m2t"
texttransformation EWS2Java (in psmSM:"http://psm_ws") {

// Por cada Puerto se genera una Facade para acceder a las operaciones del mismo ----------
  psmSM.ESStubGenerator::main(){
            self.iesfacport->forEach(esfp) { esfp.generateJavaClass() }
}

// Una vez generada la fachada, se genera el stub usando el framework AXIS ----------
psmSM.ESStubGenerator::generateAxisStub()
  {
    var wsdlAddress : String = self.wsdlAddress;

      java("adapters.wsdl2java.WSDL2JavaAdapter","generateStubfrom",wsdlAddress,"C:/Documents and
            Settings/Ricardo Quintero/Mis documentos/Doctorado/Tesis/New Tesis/Tesis/wsTogether-EMF
            MDA/JavaAdapters2AXIS/bin/")
  }
}
```

Figura 7.17 Extracto de transformación *Modelo-A-Texto* para generar servicios ajenos

Las Figuras 7.18 (a-d) muestran ejemplos de código final generado para cada paso de una clase fachada que permite acceder al servicio ajeno de `Google`. Se muestran los diferentes elementos obtenidos acorde al algoritmo de transformación anterior. El código completo se encuentra en el Apéndice 1.

```
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
public class GoogleSearchServiceGoogleSearchPortFacade
{
   private GoogleSearchService service =
       new GoogleSearchServiceLocator();
   private GoogleSearchPort port;
```

Figura 7.18(a) Ejemplo de código generado para el paso (i)

```
public GoogleSearchServiceGoogleSearchPortFacade()
throws ServiceException
{
       initPort();
}
```

Figura 7.18(b) Ejemplo de código generado para el paso (ii)

```
public void doGetCachedPage(String key, String url)
throws RemoteException
{
       port.doGetCachedPage(key,url);
}

public String doSpellingSuggestion(String key, String phrase)
 throws RemoteException
{
       return port.doSpellingSuggestion(key,phrase);
}
```

Figura 7.18(c) Ejemplo de código generado para el paso (iii)

```
private void initPort() throws ServiceException
{
       port = service.getGoogleSearchPort();
}
```

Figura 7.18(d) Ejemplo de código generado para el paso (iv)

7.2.2 Escenario 2: Generación de Servicios Propios como vistas de Clases de Negocio

Este escenario se ha implementado en cinco pasos:

(1) En el **primer paso** (ver Figura 7.19) se genera un servicio propio para una operación de una clase de negocio que se desea exponer como servicio Web para consumo por la aplicación actual (por ejemplo, desde un página Web) o por otras aplicaciones. Esta generación se implementa mediante una transformación *Modelo-A-Modelo* que recibe en su entrada la clase (y operación) del diagrama de clases y genera en su salida un servicio propio con un solo puerto y la operación a producir.

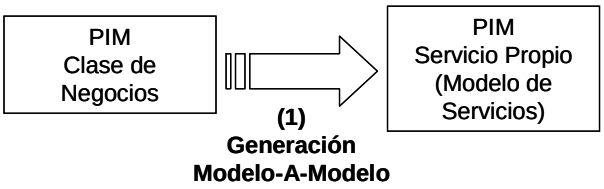


Figura 7.19 Primer paso: generación del servicio propio a partir de la clase de negocio

(2) En el **segundo paso** se genera desde el PIM de Servicio propio un par de clases PSM del Modelo de Servicios para las plataformas Java y Axis (ver Figura 7.20). La primer clase (`OSSkeletonAndWSDLGenerator`) es una clase *fabricación pura* que se utilizará para la generación de las clases *binding* para el *skeleton* y para la generación de la interfaz WSDL del servicio propio. La segunda clase (`IOSFacadePort`) permitirá la generación de una clase fachada para el acoplamiento del servicio propio con otros componentes software de la aplicación y también coadyuvará para la generación de la interfaz WSDL.

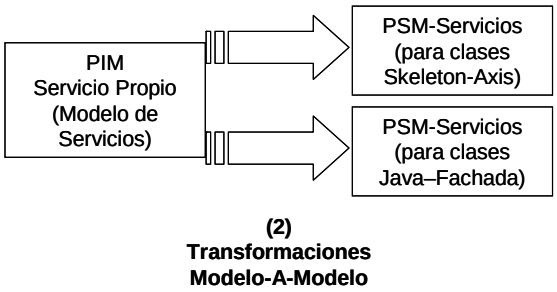


Figura 7.20 Segundo paso: generación de las clases PSM para la generación de interfaz WSDL y *Skeletons*

A partir de las clases PSM generadas, se pueden obtener todos los artefactos necesarios para la implementación del servicio propio mediante tres últimos pasos (ver Figura 7.21):

(3) En el **tercer paso** una transformación *Modelo-A-Texto* generará una clase fachada que permitirá el acceso a la operación del servicio propio. Esta clase fachada será la base para la generación del servicio Web.

(4) En el **cuarto paso**, una vez generada la clase fachada se hace uso de la herramienta `Java2WSDL` de Axis para la generación de la interfaz WSDL del servicio Web a partir de la fachada .

(5) Finalmente, en el **quinto paso** con la interfaz WSDL, y utilizando los valores almacenados en la clase `OSSkeletonAndWSDLGenerator` del PSM de servicios, se hace uso de la herramienta `WSDL2Java` para generar las clases *binding* y *skeleton* del servicio Web, así como los archivos de instalación que posibilitarán su montaje en un servidor de aplicaciones J2EE.

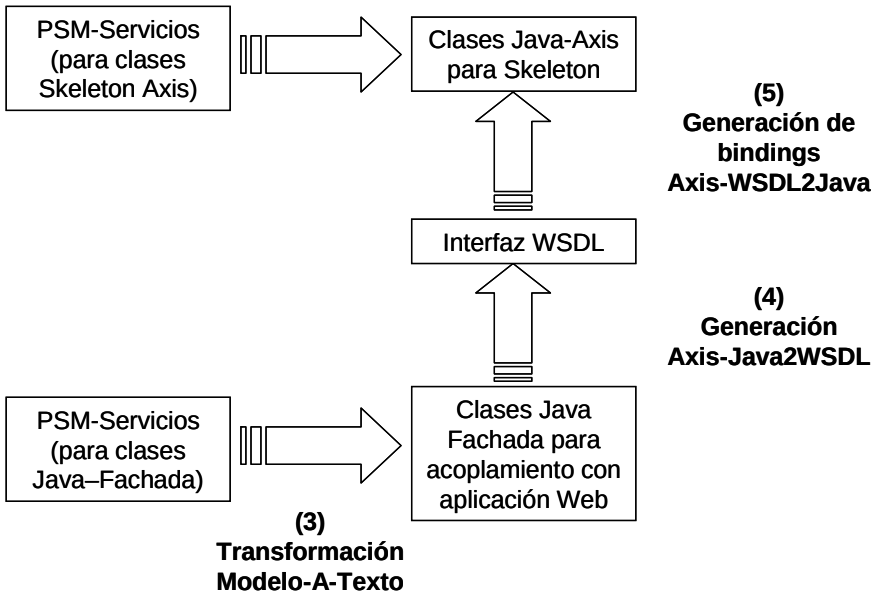


Figura 7.21 Pasos finales para la generación de servicios propios

7.2.2.1 Metamodelo PIM

El primer metamodelo PIM que participa en este escenario se refiere a las clases de negocio de la aplicación. La Figura 7.22 muestra un extracto del mismo. Las clases que se han utilizado del mismo son las siguientes:

- **Class**: una clase de negocio. Solamente se utiliza el nombre de la clase (`name`).
- **Operation**: cada clase (`Class`) puede tener una o más operaciones (`Operation`). Para este escenario se ha implementado solamente la generación de un servicio propio con operaciones *OneWay* o *RequestResponse*.
- **Parameter**: cada operación puede tener uno o más parámetros (`Parameter`). En cada uno de ellos se utiliza su nombre (`name`) y su tipo de retorno (`type`).

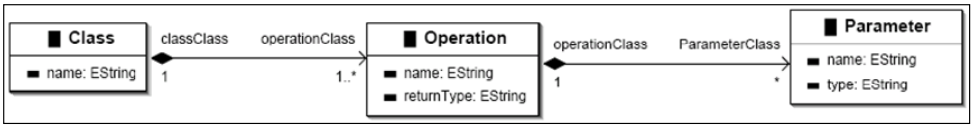


Figura 7.22 El metamodelo PIM de clases de negocio

Como ejemplo de esta clase de negocio, la Figura 7.21 muestra la clase de negocio `Producto` con las operaciones `getDescription`, `getPrice`, `setAuthor`, `setSold` y `getSold`.

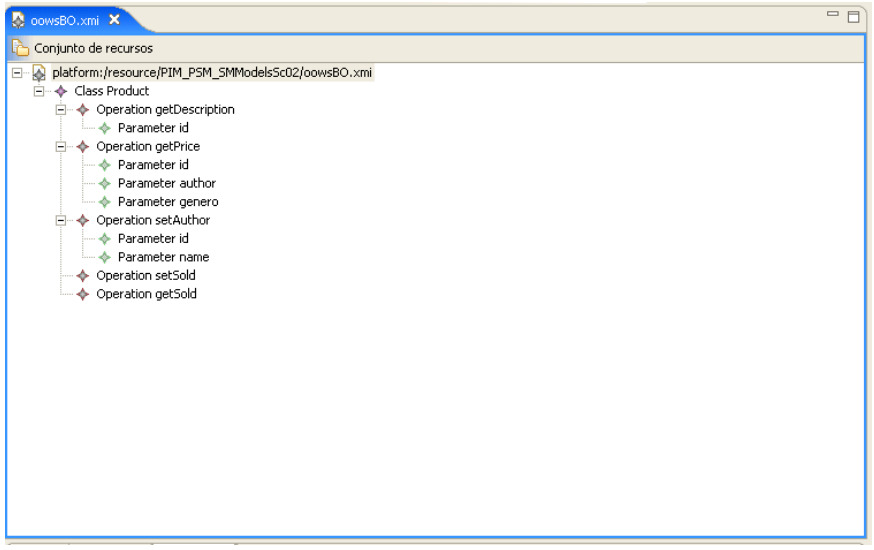


Figura 7.23 La clase de negocio `Producto`

El segundo metamodelo PIM (ver Figura 7.24) tiene la misma estructura del PIM de Servicios ajenos del escenario anterior (por lo que no se repetirá su descripción) con la diferencia de que la clase participante será el servicio propio (`OwnService`).

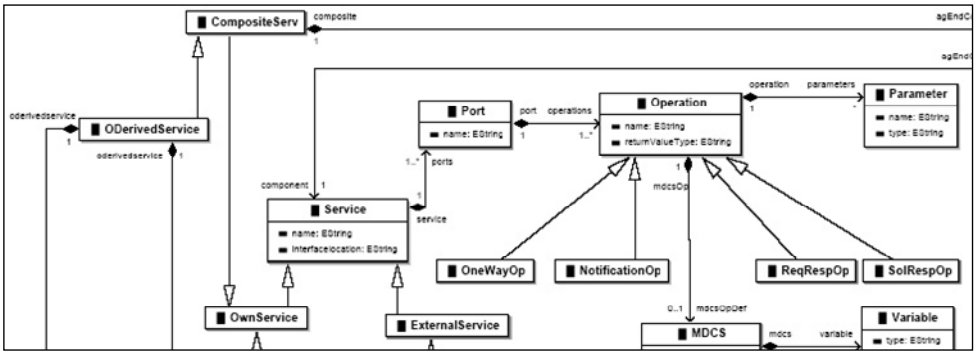


Figura 7.24 PIM de Servicios propios

7.2.2.2 Metamodelo PSM

El PSM de Servicios se muestra en la Figura 7.25. Este PSM se ha diseñado para la generación del *skeleton* y la interfaz WSDL del servicio propio. Posee las siguientes clases:

- **OSSkeletonAndWSDLGenerator**: una clase *fabricación pura* que se utiliza para la generación del *skeleton* y la interfaz WSDL. Esta clase es un repositorio de valores necesarios para su generación. Sus atributos incluyen: el nombre de la clase fachada que será expuesta como servicio Web (`classToDeploy`); una llave opcional (`key`) para el acceso al servicio propio; el nombre del paquete que contendrá la clase (`packageName`); el espacio de nombres para la interfaz WSDL (`targetNameSpace`); la dirección de la interfaz WSDL (`wsdlAddress`) y el nombre del archivo que contendrá la interfaz (`wsdlFileName`).
- **IOSFacadePort**: esta clase permite la generación de la fachada Java para acceder a la operación de la clase de negocio. Tiene solamente como atributos el nombre del puerto (`namePort`) y el nombre del servicio (`nameService`). Es una fachada con una sola operación.
- **OSOperation**: la operación de la fachada mediante la cual se accede a la operación de la clase de negocio. Posee dos atributos: el nombre de la operación (`name`) y el tipo de retorno (`returnValueType`).
- **OSParameter**: los parámetros de la operación. Posee dos atributos: el nombre del parámetro (`name`) y el tipo de retorno del parámetro (`type`).

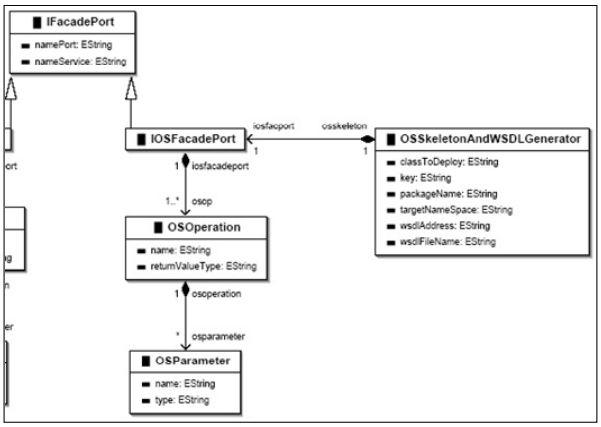


Figura 7.25 PSM de Servicios propios

El PSM de Servicios propios para la clase `Producto` se muestra en la Figura 7.26.

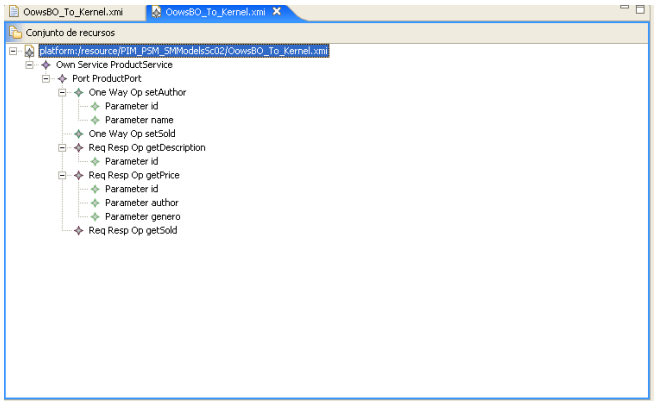


Figura 7.26 El PSM de Servicios propios para la clase `Producto`

7.2.2.3 Reglas de transformación PIM-A-PIM

La generación del servicio propio a partir de la clase de negocio del diagrama de clases se lleva a cabo mediante el siguiente conjunto de reglas de transformación (ver Figura 7.27):

- **makeServiceAndPort**: desde la clase de negocio (`Class`) permite generar el servicio propio (`OwnService`). El nombre del servicio (`Service.name`) se obtiene a partir del nombre de la clase de ne-

gocio (`Class.name`). Solamente se genera un puerto (`Port`) con todas las operaciones de la clase de negocio.

- **makeOperationOneWay**: si la operación de negocio tiene como valor de retorno el tipo `void` esta regla la corresponderá con una operación *OneWay*.
- **makeOperationReqResp**: si la operación de negocio tiene como valor de retorno un tipo diferente a `void` esta regla de transformación generará una operación *Request-Response* (`ReqRespOp`).
- **makeParameter**: para cada operación generada (de cualquiera de los dos tipos mencionados), esta regla de transformación genera sus parámetros.

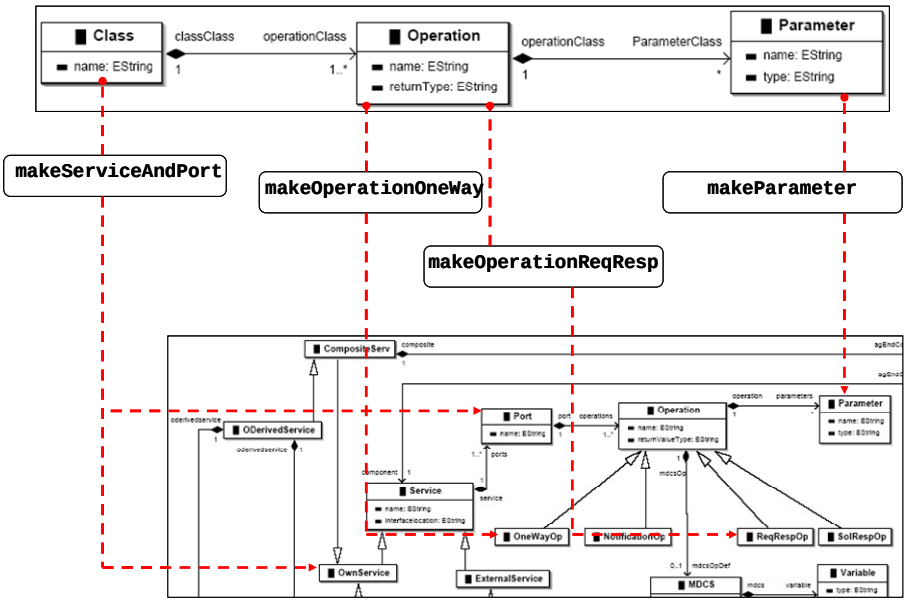


Figura 7.27 Reglas de transformación PIM-A-PIM para la generación del servicio propio a partir de la clase de negocio

La Figura 7.28 muestra la implementación de la regla de transformación `makeServiceAndPort`. Esta regla se recibe en su entrada la clase de negocio (`oowsBO`) y obtiene en su salida el servicio propio generado (`kernel::OwnService`). La regla asigna el nombre (`name`) del servicio propio concatenado con el sufijo `'Service'` y construye los puertos (`ports`) con las transformaciones restantes.

La implementación de otras transformaciones se encuentra en el Apéndice 2.

```
/* De cada clase OOWS-BO (PIM) se crea 1 Servicio propio con 1 puerto --------- */
mapping makeServiceAndPort(in model: oowsBO::Class): kernel::OwnService {
init {
var allOperationsOneWay  := model.operationClass->select(o|o.returnType =  'void');
var allOperationsReqResp := model.operationClass->select(o|o.returnType <> 'void');
}
object {
         name := model.name + 'Service';
         ports +=
         object kernel::Port
         {
                  name := model.name + 'Port';
                  operations += allOperationsOneWay->
                  collect(o|makeOperationOneWay(o));
                  operations += allOperationsReqResp->
                  collect(o|makeOperationReqResp(o));
         }
         }
}
```

Figura 7.28 Regla de transformación `makeServiceAndPort` implementada en QVT-operacional

7.2.2.4 Reglas de transformación PIM-A-PSM

Una vez que se ha generado el servicio propio, éste se transforma al PSM de Servicios que posibilitará después la generación de su *skeleton* e interfaz WSDL. El conjunto de transformaciones necesario para obtener estos artefactos es el siguiente (ver Figura 7.29):

- **makeSkeleton**: partiendo del servicio propio (`OwnService`) se genera la clase *fabricación pura* `OSSkeletonAndWSDLGenerator`, un repositorio de valores para la generación del *skeleton* y la interfaz WSDL del servicio propio.
- **makeOperation**: permite la generación de las operaciones del servicio propio a partir de las operaciones del PIM de servicios. Se aplica tanto para las operaciones *OneWay*, como las operaciones *Request-Response* que fueron generadas en el paso de transformación anterior.
- **makeParameter**: transforma los parámetros de las operaciones del PIM de servicios a los parámetros de las operaciones del servicio propio a producir.

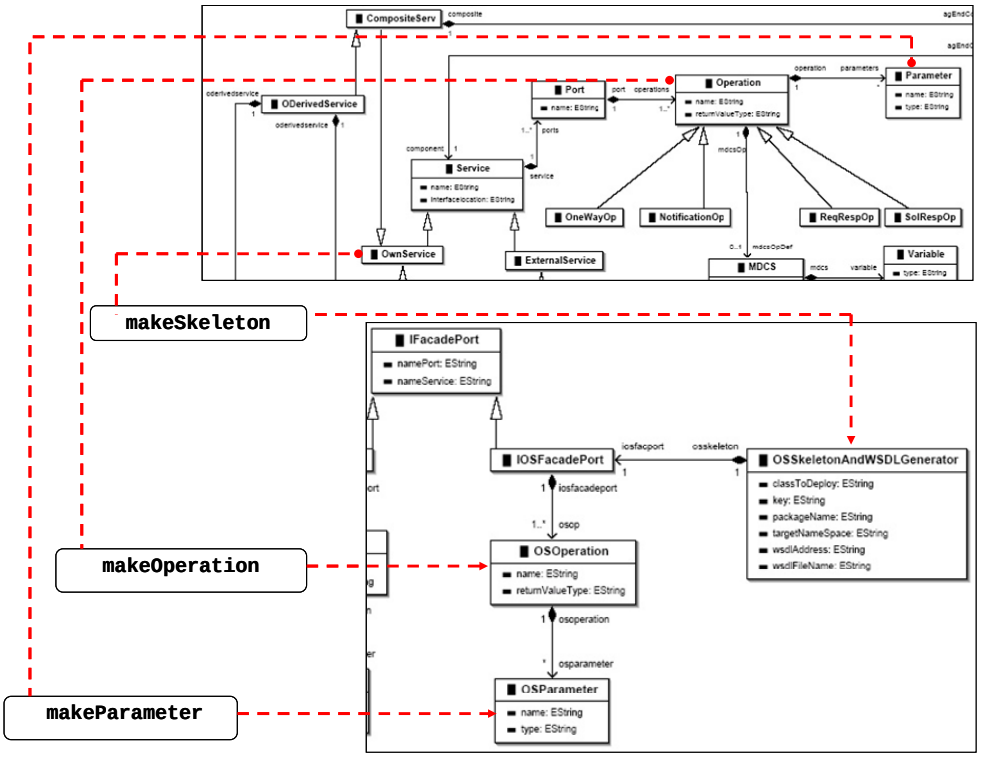


Figura 7.29 Transformaciones del PIM de Servicios al PSM de Servicios

7.2.2.5 Reglas de transformación PSM-A-Código

La generación de código final se realiza mediante **dos pasos**:

(1) Una transformación *Modelo-A-Texto* para la generación de una clase fachada de las operaciones de la clase de negocio.

(2) La generación del *skeleton* y la interfaz WSDL a partir de la clase fachada anterior y los valores que han sido almacenados en la clase *OSSkeletonAndWSDLGenerator* en el paso anterior.

El algoritmo de generación de código Java y Axis está dado por los siguientes pasos:

1. Genera, a partir de la clase *IOSFacade* del PSM de servicios, la clase fachada que será expuesta como servicio propio:

a. Genera el inicio de la clase.
b. Genera los atributos de infraestructura de la clase.
    i. Genera el atributo `businessObject` para referenciar al objeto de negocio.
    ii. Inicializa el atributo `businessObject` con una referencia al objeto de negocio.
c. Para cada operación del objeto de negocio.
    i. Genera el tope de la operación (su signatura).
    ii. Genera el código de delegación a la operación del objeto de negocio.

2. Genera la interfaz WSDL para la clase Java-Fachada mediante la herramienta `Java2WSDL` de Axis.
3. Genera el código *skeleton* a partir de *OSSkeletonAndWSDLGenerator* y la interfaz WSDL mediante la herramienta `WSDL2Java` de Axis.

Finalmente la instalación del servicio propio en un servidor de aplicaciones J2EE se realiza ejecutando el fichero de instalación `deploy.wsdd` generado en el último paso del algoritmo.

La Figura 7.30 muestra parte de la implementación de esta transformación para la generación del código de las operaciones de la fachada. El código incluye la generación del tope (`signature`) de la operación. Éste se obtiene a partir del tipo de retorno (`self.returnValueType`), el nombre (`self.name`) y la lista de parámetros (`self.osparameter`).

La implementación de la transformación completa se encuentra en el Apéndice 2.

```
texttransformation OSOperation2Java (in psmSM:"http://psm_ws") {

// Generando una operacion de la Facade -------------------------

psmSM.OSOperation::generateJavaOp()
{
  self.generateSignature();
  self.generateBody();
}

// Se genera el signature de la operacion -----------------------

psmSM.OSOperation::generateSignature()
{
  var nParams: Integer = self.osparameter.size();

  'public ' self.returnValueType ' ' self.name
  '('
  if (nParams > 0)
  {
        self.osparameter.first().generateParameter();
                        self.osparameter->forEach(p)
  {
        if (position() >= 1)
        {
           ', ' p.generateParameter()
        }
  }
  }
  ')\n'
}
```

Figura 7.30 Extracto de la Transformación *PSM-A-Código* para el escenario 2

La Figura 7.31 muestra un extracto de la clase Java fachada que se ha generado a partir del PSM de Servicios. Se observa el atributo `Product` que hace referencia al objeto de negocio que será expuesto como servicio Web. También se muestra la delegación de llamada a una de las operaciones del mismo (`setAuthor`).

```
public class ProductServiceProductPortFacade
{
  private Product bo = new Product();
  public void setAuthor(String id, String name)
  {
       bo.setAuthor(id,name);
  }
//...
}
```

Figura 7.31 Extracto de una clase Java-fachada para la producción de un servicio propio.

Es a partir de esta clase Java fachada que, utilizando la herramientas Axis mencionadas, se genera el código *skeleton* e interfaz WSDL que permitirá su instalación en el servidor de aplicaciones para su posterior consumo.

7.2.3 Escenario 3: Agregación y Composición de Servicios

En el escenario de generación de código para la Agregación y Composición de Servicios se recibe en su entrada un Modelo de Servicios con la definición de un servicio compuesto que agrega uno o más servicios componente. Además, la lógica de cada operación del servicio compuesto se define mediante un MDCS. El escenario obtiene en su salida el código para la implementación WS-BPEL del servicio compuesto.

El escenario se ha implementado en cinco pasos:

(1) En el **primer paso** (ver Figura 7.32) se inicia con el Modelo de Servicios y el MDCS para generar un modelo PSM-BPEL que sólo posee las variables definidas por el usuario sin incluir las correspondientes a los mensajes de entrada/salida para llamadas a operaciones de servicios componente.

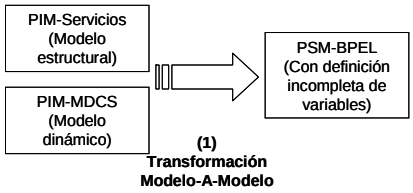


Figura 7.32 Generación de PSM-BPEL con definición incompleta de variables

(2) En el **segundo paso** (ver Figura 7.33) el PSM-BPEL con definición incompleta de variables se transforma en un PSM-BPEL con definición completa de variables. Esto se logra agregando las variables para los mensajes de entrada/salida necesarios para las llamadas a las operaciones de los servicios componente.

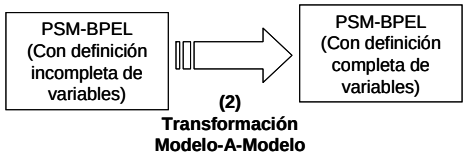


Figura 7.33 El PSM-BPEL completado con variables para mensajes de entrada/salida faltantes

(3) En el **tercer paso** (ver Figura 7.34), a partir del PSM-BPEL con definición completa de variables, se obtienen los artefactos necesarios para la instalación en un entorno de ejecución para cada proceso BPEL que implementa cada operación del servicio compuesto. Para el entorno seleccionado, *Oracle BPEL Process*, las transformaciones *Modelo-A-Texto* generan: el proceso BPEL, su interfaz WSDL, una interfaz envolvente (*Wrapper*) para las interfaces WSDL de los servicios componente, el descriptor de procesos, el descriptor y el constructor del proyecto

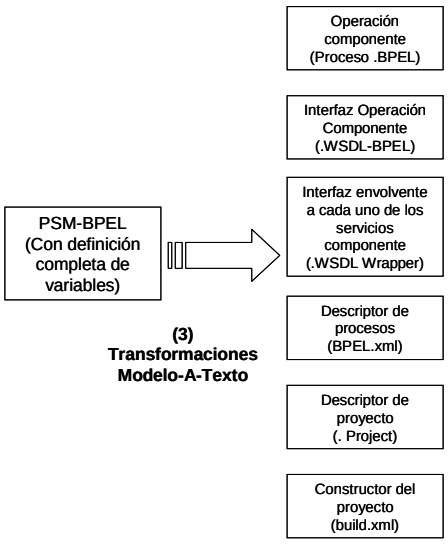


Figura 7.34 Generación de artefactos a partir del PSM-BPEL para el entorno de ejecución

(4) El **cuarto paso** (ver Figura 7.35) es una transformación *Modelo-A-Texto* que genera dos tipos de fachada: (1) una que ofrece acceso a cada operación compuesta implementada en BPEL y (2) otra que ofrece acceso único a todas las fachadas de todas las operaciones del servicio compuesto.

(5) El **quinto paso** utiliza la herramienta `wsdl2java` de Axis para la generación del código *stub* necesario para el funcionamiento del primer tipo de fachada del cuarto paso.

(6) Finalmente, en el **sexto paso** se utiliza la herramienta `java2wsdl` para generar el código *skeleton* necesario para la instalación del servicio compuesto, así como también la interfaz WSDL para su consumo por otras aplicaciones.

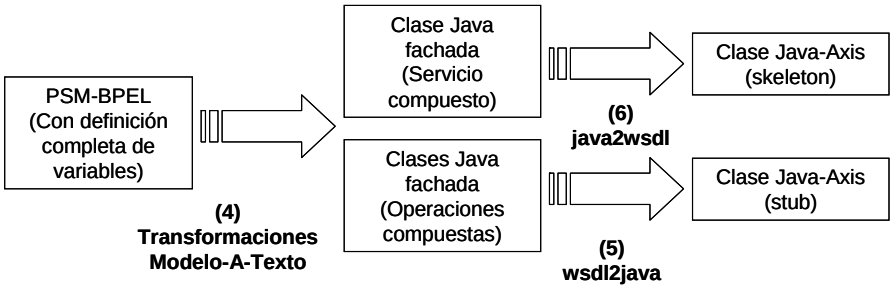


Figura 7.35 Generación de clases Java-fachada para servicio compuesto y operaciones componente

7.2.3.1 Metamodelos PIM

En este escenario se utiliza el PIM de Servicios con sus modelos estructural y dinámico. En lo que se refiere al modelo estructural (ver Figura 7.36), básicamente se utilizan las mismas clases para servicios propios y ajenos que ya han sido explicadas en los escenarios anteriores más un conjunto de clases para definir las relaciones estructurales entre servicios. Es importante resaltar que el servicio compuesto siempre será un servicio propio y los servicios componente pueden ser propios o ajenos.

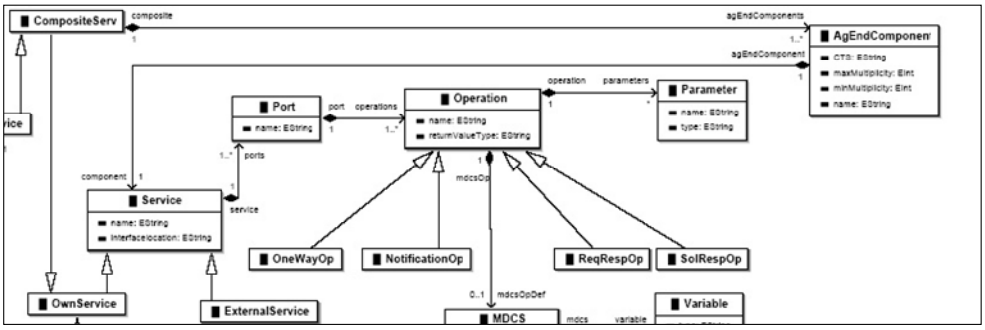


Figura 7.36 El PIM de servicios para el modelado estructural de servicios compuestos

Las clases que se utilizan para el modelado estructural son las siguientes:

- **OwnService**: servicio propio componente. Sus atributos ya han sido explicados en escenarios anteriores.

- **CompositeService**: servicio propio compuesto que se desea definir. Es un servicio propio (`Ownservice`).
- **ExternalService**: servicio ajeno que podrá ser utilizado como servicio componente. Para su uso dentro de este escenario se requerirá un paso de importación *Texto-A-Modelo* semejante al que fue explicado en el escenario 1. Sus atributos ya han sido explicados en escenarios anteriores.
- **Port**: se implementa un solo puerto para el servicio compuesto. Ya ha sido explicado anteriormente.
- **Operation**: podrán utilizarse los cuatro tipos de operaciones en los servicios componente. Para el servicio compuesto solamente se han implementado las operaciones *OneWay* y *Request-Response*.
- **Parameter**: los parámetros de las operaciones de los servicios componente y compuesto.
- **AgEndComponent**: define la forma en que será utilizado el servicio componente propio o ajeno. El capítulo 4 ha explicado su uso mediante el uso del *framework* multidimensional.

La Figura 7.37 muestra el PIM-MDCS que permite definir la lógica de cada operación compuesta.

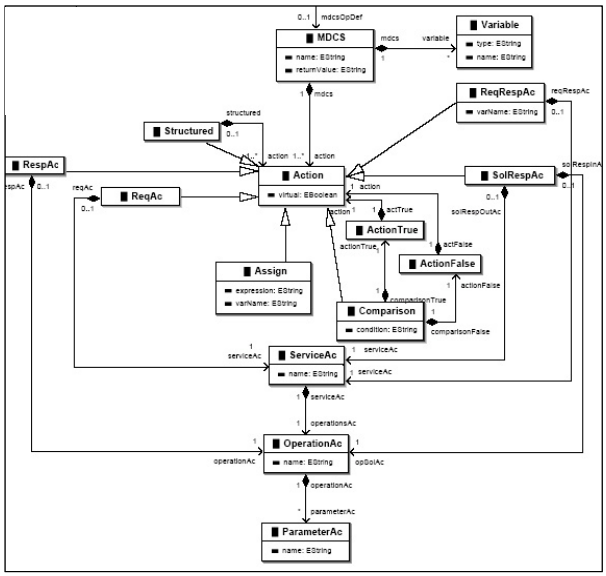


Figura 7.37 El PIM-MDCS para el modelado dinámico de composición de servicios

Las clases de este PIM son las siguientes:

- **MDCS**: clase principal para la definición de la lógica de una operación compuesta. Posee dos atributos: el nombre de la operación (`name`), que dará origen al nombre del proceso de implementación y el valor de retorno de la misma (`returnValue`).
- **Variable**: cada operación compuesta puede definir cero o más variables locales. Cada variable local puede ser especificada mediante dos atributos: su nombre (`name`) y su tipo (`type`).
- **Action**: una acción de la operación compuesta. Posee el atributo (`virtual`) para indicar su posible redefinición (en el escenario de Especialización de Servicios). Esta clase se especializa en los siguientes subtipos:
    - **Structured**: que permite el agrupamiento de un conjunto de acciones. No posee atributos.
    - **ReqAc**: permite invocar una operación *OneWay* de un servicio componente. La definición completa de la llamada se establece referenciando al servicio que se invoca (`ServiceAc`), operación invocada (`OperationAc`) y sus parámetros (`ParameterAc`).
    - **RespAc**: permite implementar la operación de notificación (*Notify*). Se debe especificar la operación (`OperationAc`) y sus parámetros (`ParameterAc`).
    - **ReqRespAc**: permite invocar una operación *Request-Response* de un servicio componente especificando además el servicio (`ServicAc`), operación (`OperationAc`) y parámetros (`ParameterAc`) invocados. Posee el atributo `varName` para indicar la variable donde se recibirá el resultado de la invocación, esta variable debió ser declarada con anterioridad en el modelo.
    - **SolRespAc**: permite recibir una solicitud de un cliente, indicando la operación (`OperationAc`) y sus parámetros (`ParameterAc`); así como enviar una respuesta al cliente invocando una operación *OneWay* del mismo, especificada

por su servicio (`ServiceAc`), operación (`OperationAc`) y parámetros (`ParameterAc`).

- **Assign**: permite la definición de una acción de asignación de una expresión a una variable. La expresión se define en el atributo `expression` y la variable en el atributo `varName`.
- **Comparison**: permite la definición de una acción de control condicional. La condición se define en el atributo `condition` y las acciones a ejecutar, dependiendo del valor lógica de la condición, se definen en las clases *ActionTrue* y *ActionFalse*.

Como ejemplo de este PIM se muestra en la Figura 7.38 un extracto del servicio compuesto `BestStoreService` expuesto en el capítulo 4.

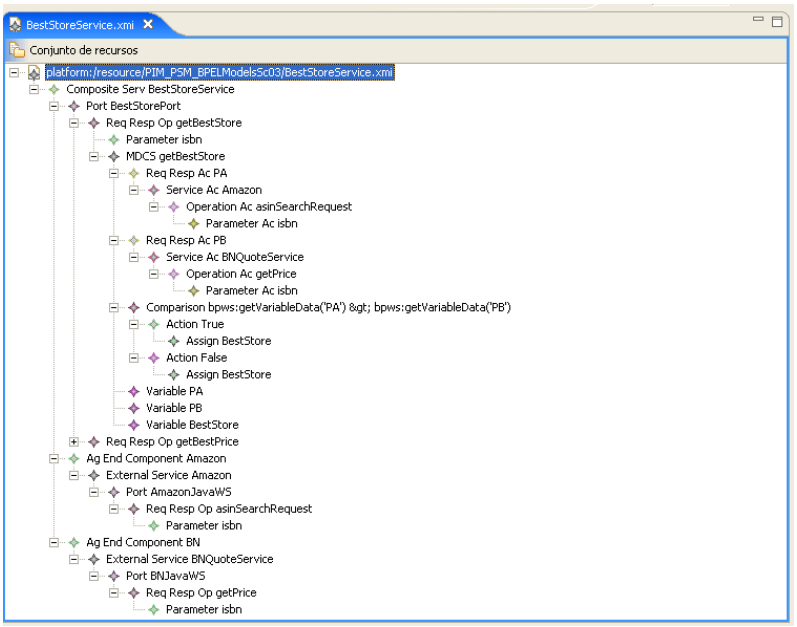


Figura 7.38 Ejemplo de servicio compuesto `BestStoreService`

7.2.3.2 Metamodelo PSM

La Figura 7.39 muestra las clases del metamodelo PSM que se han definido para la generación de las fachadas Java-Axis para las operaciones y el servicio compuesto.

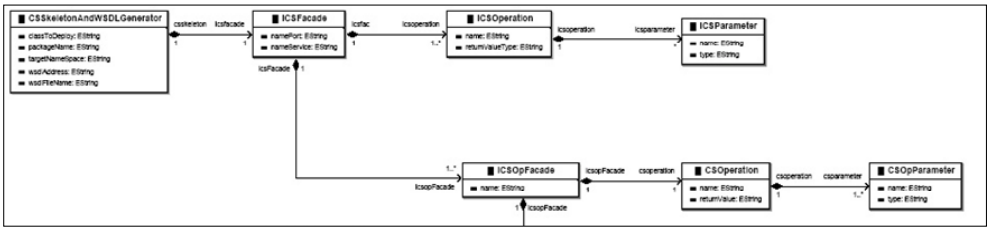


Figura 7.39 Clases del metamodelo PSM-BPEL para la generación de fachadas

Las clases utilizadas para generar las fachadas son las siguientes:

- **CSSkeletonAndWSDLGenerator**: es semejante a la clase *OSSkeletonAndWSDLGenerator* para el escenario de generación de servicios propios: una clase *fabricación pura* cuyos atributos permiten la generación de la clase fachada para el servicio propio compuesto utilizando la herramienta `java2wsdl`.
- **ICSFacade**: cada objeto *CSSkeletonAndWSDLGenerator* se asocia con un objeto de esta clase la cual permite la generación de la fachada del servicio propio. Sus dos atributos (`namePort` y `nameService`) se utilizan de manera combinada para la definición del paquete de la clase Java y su nombre.
- **ICSOperation**: cada objeto *ICSFacade* se asocia con uno o más objetos de esta clase. Se utiliza para la generación de las signaturas de las operaciones compuestas. Posee dos atributos: el nombre de la operación (`name`) y su tipo de retorno (`returnValueType`).
- **ICSParameter**: define los parámetros de cada operación. Posee dos atributos: su nombre (`name`) y su tipo (`type`).
- **ICSOpFacade**: esta clase permite la generación de una o más fachadas para las operaciones de servicios compuestos. Posee como

atributo el nombre (`name`) de la operación que se está implementando.

- **CSOperation**: la fachada de la operación poseerá solamente una operación que se define con esta clase. Posee el nombre (`name`) y el valor de retorno (`returnValue`).
- **CSOpParameter**: los parámetros de las operaciones definidos mediante los atributos de su nombre (`name`) y tipo (`type`).

Además del metamodelo para la plataforma Java y Axis se ha definido también un metamodelo PSM para la implementación de las operaciones del servicio compuesto (ver Figura 7.40). Cada operación se implementa como un proceso BPEL. El metamodelo representa un subconjunto de la plataforma incluyendo sólo las primitivas necesarias para la implementación de las operaciones.

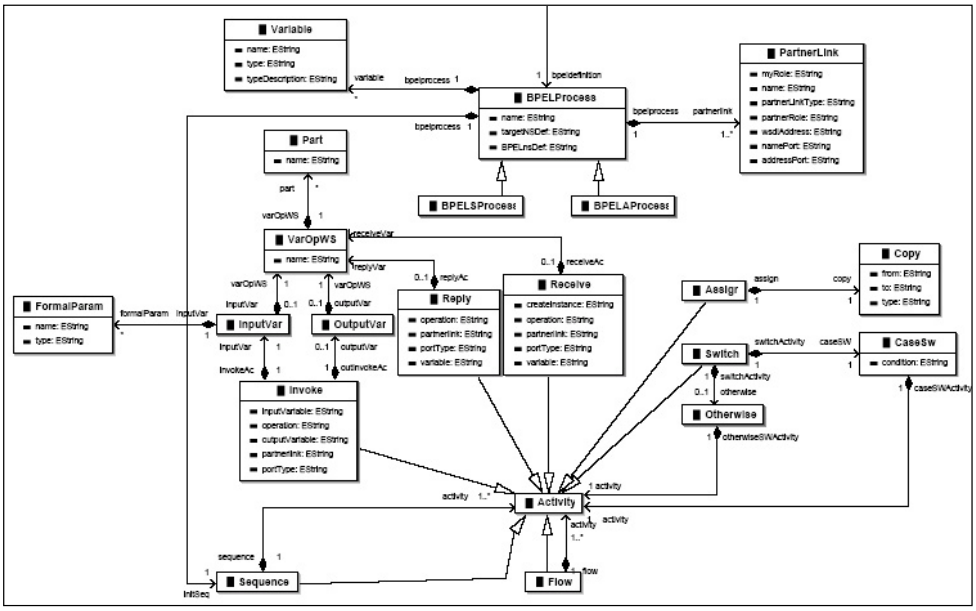


Figura 7.40 El metamodelo PSM-BPEL

Cada una de las clases del metamodelo se explica a continuación:

- **BPELProcess**: clase inicial para la definición del proceso BPEL. Posee tres atributos: el nombre del proceso (`name`). El espacio de

nombre destino para el archivo XML (`targetNSDef`) y el espacio de nombres para la definición de los procesos BPEL (`BPELnsDef`). La clase se puede especializar para la definición de procesos síncronos (*BPELSProcess*) y asíncronos (*BPELAProcess*).

- **PartnerLink**: en esta clase se definen los enlaces a los servicios componente del proceso. Los atributos que caracterizan estos enlaces son: el nombre del enlace (`name`); el rol que juega el proceso actual (`myRole`); el rol que juega el servicio componente (`partnerRole`); el tipo del enlace (`partnerLinkType`) que se define en las clases envolvente (*Wrapper*); la dirección URL de la interfaz del servicio componente (`wsdlAddress`); el nombre del puerto que se está utilizando en el servicio componente (`namePort`) y la dirección URL del puerto (`addressPort`).
- **Variable**: la definición de las variables que se utilizarán en el proceso BPEL. Dicha definición se caracteriza por los siguientes atributos: nombre de la variable (`name`); tipo de la variable –*messageType*, *element*, *type*- (`type`) y descripción del tipo XML Schema de la variable (`typeDescription`).
- **Activity**: cada una de las actividades del proceso BPEL. Esta clase se especializa en las siguientes actividades:
    - **Sequence**: que puede presentarse en el proceso en dos formas: como la actividad inicial principal del mismo, la cual contiene a todas las actividades del mismo; o bien como una actividad más que permite definir una actividad compuesta por otras actividades.
    - **Invoke**: permite invocar a un servicio componente de forma síncrona (para una operación *Request-Response*) o asíncrona (para una operación *OneWay*). Utiliza los siguientes atributos: el enlace al servicio componente (`partnerlink`); la operación a invocar (`operation`); el mensaje de entrada (`inputVariable`) y de salida (`outputVariable`). Estos mensajes son definidos mediante un par de variables de entrada y salida que fueron generadas automá-

ticamente en el paso de transformación anterior. Las variables se especifican mediante las clases *VarOpWS* y *Part* (siguiendo la especificación XPath [59]). Finalmente se guarda, para efecto de generación posterior de código, también el nombre del parámetro formal (*FormalParam*) que dio origen al mensaje de la variable de entrada.

- **Reply**: permite especificar la respuesta del proceso. Incluye como atributos el nombre de la operación (`operation`); el enlace al cliente de la operación (`partnerLink`); el puerto del proceso (`portType`) y la variable de respuesta (`variable`).
- **Receive**: permite especificar la recepción del mensaje de inicio al proceso. Este mensaje se define con el atributo `variable` y está especificado con la clase *VarOpWS*, cuyas partes (`Part`) están dadas por los parámetros de la operación. Posee además los siguientes atributos: para la creación de una instancia del proceso en el motor de ejecución BPEL (`createInstance`); el nombre de la operación (`operation`); el nombre del enlace (`partnerLink`) y el nombre del puerto (`portLink`).
- **Assign**: para la definición de una operación de asignación BPEL. La asignación se define en la clase *Copy* asociada a la misma, la cual posee dos atributos: la expresión -o variable- que se está asignando (`from`) y la variable asignada (`to`).
- **Switch**: para la definición de estatutos condicionales de control. La condición del estatuto se define en el atributo `condition` y las actividades a ejecutar, dependiendo de la evaluación de su valor de verdad (clases *CaseSw* y *Otherwise*), se definen con una asociación a la clase *Activity*.

7.2.3.3 Reglas de transformación PIM-A-PSM

El conjunto de reglas de transformación que corresponden los PIM de servicios estructural y dinámico para la generación de fachadas Java-Axis posterior se definen a continuación (ver Figura 7.41):

- **makeSkeleton**: corresponde el servicio compuesto (*CompositeServ*) con el generador de skeleton e interfaz WSDL (*CSSkeletonAndWSDLGenerator*). A partir del nombre del servicio (`CompositeServ.name`) y el nombre del puerto (`CompositeServ.Port.name`) se inicializan los atributos del generador (el nombre de la interfaz y dirección WSDL, el espacio de nombres destino, el nombre del paquete y la clase Java). También se inicializa la clase *ICSFacade* que posteriormente generará la clase fachada al servicio compuesto.
- **makeOperation**: a partir de las operaciones del servicio compuesto (*Operation*) genera las operaciones de la fachada (*ICSOperation*).
- **makeParameter:** genera los parámetros de las operaciones de la fachada (*ICSParameter*) a partir de los parámetros de las operaciones del servicio compuesto (*Parameter*).
- **makeFacOperation:** genera para cada una de las operaciones del servicio compuesto (*Operation*) la fachada (*ICSOpFacade*) que será utilizada por la fachada del servicio compuesto. También a partir del nombre de la operación (`Operation.name`) se genera el nombre de la fachada (`ICSOpFacade.name`) y a partir del valor de retorno (`Operation.returnValueType`) se genera el valor de retorno de la operación en la fachada (`ICSOpFacade.CSOperation.returnValueType`).
- **makeFacParameter**: genera los parámetros (*CSOpParameter*) para las operaciones de la fachada a partir de los parámetros de las operaciones componente (*Parameter*).

La Figura 7.42 muestra la implementación de la regla de transformación `makeSkeleton`. Esta regla es la primera que se ejecuta y a partir de la misma se ejecutan las restantes. La regla corresponde el servicio compuesto (`ker-`

nel::CompositeServ) con la clase generadora de skeleton (psm_bpel::CSSkeletonAndWSDLGenerator) y va obteniendo cada uno de los atributos a partir de los atributos del servicio compuesto: el nombre de la interfaz WSDL (wsdlFileName) a partir del nombre del servicio compuesto (serviceName) concatenado con la extensión .wsdl; la dirección de la interfaz WSDL (wsdlAddress); el espacio de nombres destino (targetNameSpace) y el nombre de la clase fachada (classToDeploy) a partir del nombre del servicio compuesto (serviceName) y finalmente la definición del cuerpo de la fachada (icsfacade) a partir de las transformaciones restantes.

La implementación de otras transformaciones está dada en el Apéndice 3.

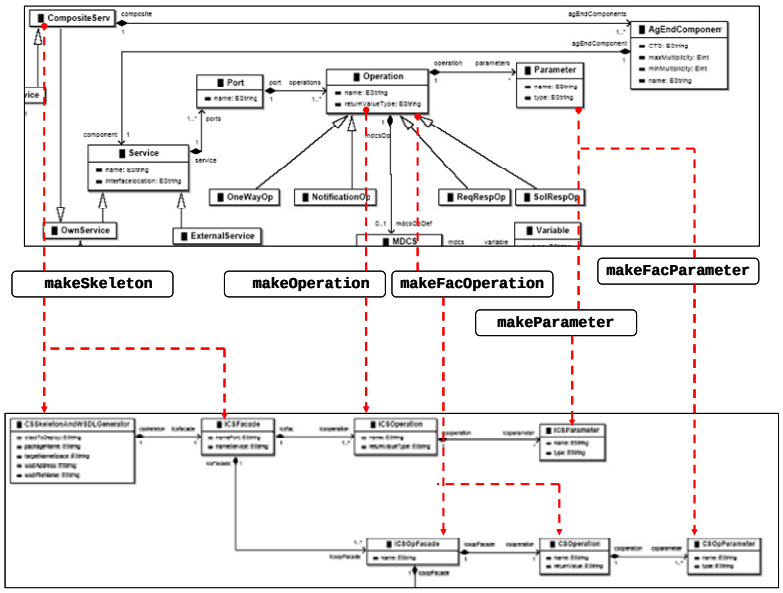


Figura 7.41 Reglas de transformación para generar fachadas para el servicio compuesto

```
transformation Kernel_To_Psm_bpelSc03;
metamodel 'http://kernel';
metamodel 'http://psm_bpel';
mapping main(in model: kernel::CompositeServ):
psm_bpel::CSSkeletonAndWSDLGenerator {
        init {
                result := makeSkeleton(model);
        }
        object {
        }
}
mapping makeSkeleton(in model: kernel::CompositeServ):
psm_bpel::CSSkeletonAndWSDLGenerator {
        init {
                var firstPort     := model.getFirstPort();
                var allOperations := firstPort.operations;
                var serviceName   := model.name+firstPort.name;
        }
        object {
                wsdlFileName    := serviceName+'.wsdl';
                wsdlAddress     := 'http://localhost:8080/axis/services/'
                                    + serviceName;
                targetNameSpace := serviceName + 'Ns';
                packageName:= serviceName + 'Pck';
                classToDeploy   := serviceName+'.java';
                icsfacade      := object psm_bpel::ICSFacade {
                        namePort := model.getFirstPort().name;
                        nameService:= model.name;
                        icsoperation += allOperations->
                                        collect(o|makeOperation(o));
                        icsopFacade  += allOperations->
                                        collect(o|makeFacOperation(o))
        }
}
```

Figura 7.42 Regla de transformación `makeSkeleton` en QVT-operacional

Las reglas básicas de transformación para la generación de la implementación de la lógica de cada una de las operaciones compuestas están dadas en las Figura 7.43 y 7.44. Se explica a continuación las reglas para inicializar el proceso BPEL (Figura 7.43):

- **makeBPELProcess**: que permite la inicialización del proceso BPEL (*BPELProcess*). A partir del nombre de la definición del MDCS (`MDCS.name`) se inicializa el nombre del proceso. El espacio de nombres destino (`targetNSDef`) se inicializa a partir del nombre del servicio compuesto (`Service.name`). El espacio de nombre para el proceso BPEL (`BPELnsDef`) se define de manera constante.

- **makeBPELvariable**: la cual define una variable de entrada (`input`) y de salida (`output`) para los parámetros de entrada de la operación y el valor de retorno del proceso. Además corresponde todas las definiciones de variables locales creadas por el usuario (*Variable*) con variables del proceso BPEL (*Variable*). Faltarían las variables de entrada y salida necesarias para la invocación de servicios Web que se obtendrán en un proceso de transformación posterior.

- **makePartnerLink**: esta regla se aplica a cada uno de los servicios componente para generar un *PartnerLink* BPEL. Se parte de los extremos finales de los servicios componente (*AgEndComponent*) y se generan los *PartnerLink* del proceso BPEL.

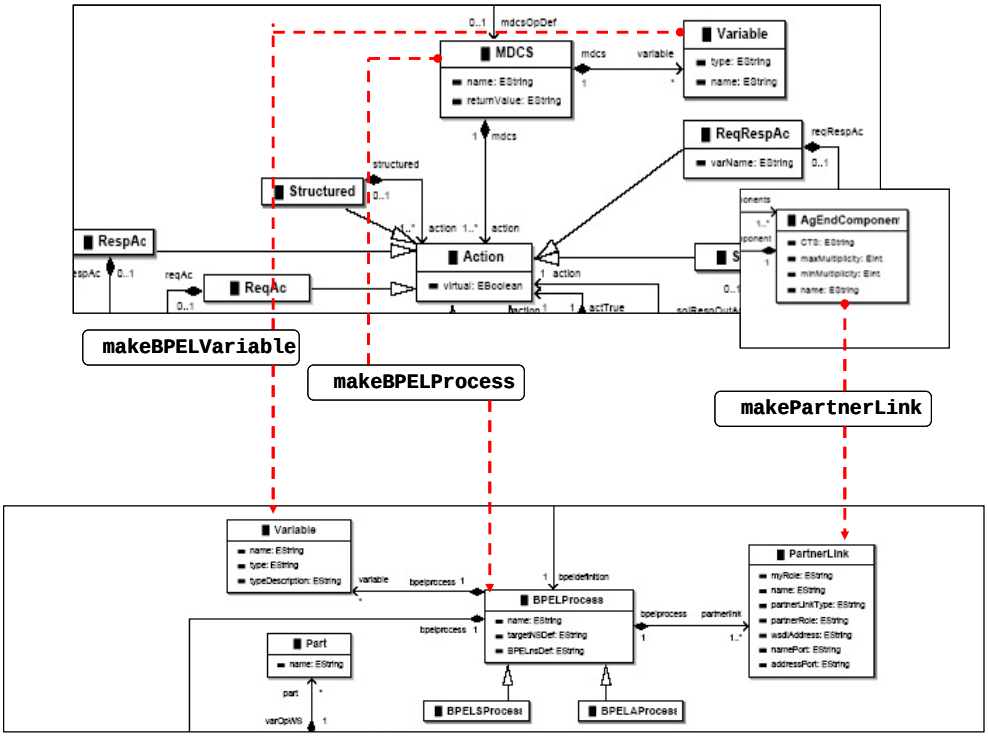


Figura 7.43 Reglas de transformación para la inicialización del proceso BPEL

Una vez inicializado el proceso BPEL el siguiente conjunto de reglas de transformación mapea las acciones del MDCS hacia las actividades BPEL que la implementan (ver Figura 7.44):

- **makeSequence**: a partir una acción *Structured* genera una actividad *Sequence*. Al momento de la inicialización del proceso BPEL se genera una actividad *Sequence* por defecto con una actividad *Receive* al inicio para recibir el mensaje de entrada al proceso (el cual contendrá los parámetros de entrada al proceso) y una actividad *Reply* al final para devolver la respuesta de la operación.

- **toInvoke (desde ReqAc)**: que permite corresponder una acción *Request* con una actividad *Invoke* BPEL. Inicializa sus atributos, incluyendo la variable de entrada (`inputVar`) y sus partes (`varOpWS`), con los parámetros actuales de la invocación de la operación.
- **toInvoke (desde ReqRespAc)**: es semejante a la regla anterior solo que aquí se corresponde una acción *Request-Response* con una actividad *Invoke*. Además se genera la variable de salida (`outputVar`) para la respuesta de la invocación.
- **toReceive** : que mapea una acción *RespAc* con una actividad *Receive*, inicializa además los parámetros de entrada (`receiveVar`).
- **toAssign**: que corresponde una acción de asignación (*Assign*) con una actividad de asignación BPEL (*Assign*). Las expresiones asignadas se compilan para su transformación en expresiones *XPath* que puede manejar BPEL.
- **toReceiveInvoke**: que corresponde una acción *Solicit-response* con un par de actividades BPEL generadas: *Receive*, para recibir la solicitud del cliente e *Invoke*, para entregar la respuesta.
- **toSwitch**: que permite la generación de una actividad *Switch* (con sus clases *Otherwise* y *CaseSw*) a partir de una acción *Comparison*.

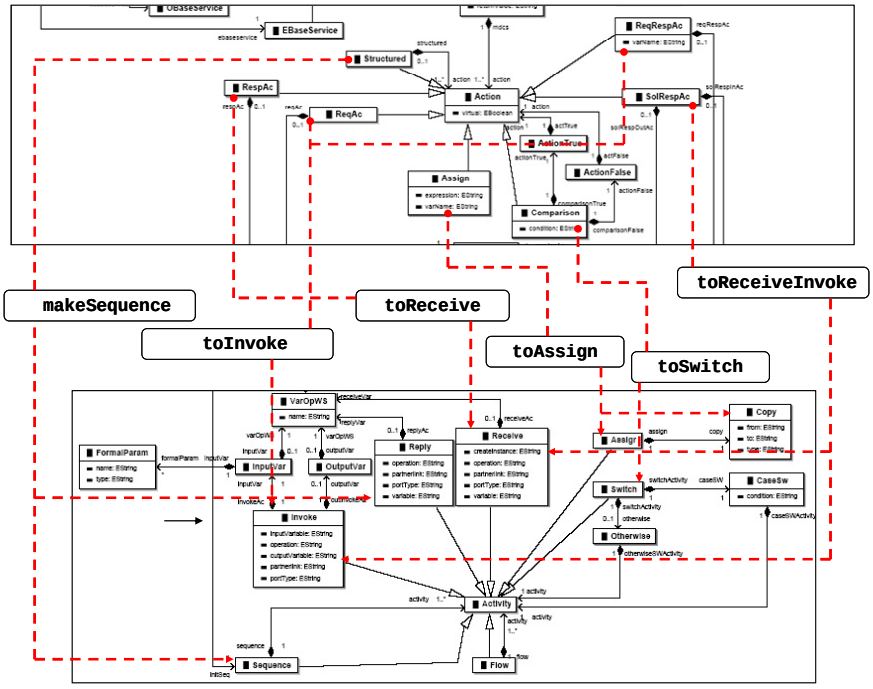


Figura 7.44 Reglas de transformación para la generación de la lógica de la operación compuesta

La Figura 7.45 muestra un extracto de la regla `makeFacOperation` que incluye la sección de generación del código BPEL desde la cual se invocan todas las reglas de generación de actividades BPEL anteriormente explicadas. Al inicio la regla genera una actividad *Sequence* (`psm_bpel::Sequence`) que contendrá la lógica de la operación. Luego dentro del *Sequence* genera una actividad *Receive* (`psm_bpel::Receive`) para recibir el mensaje de entrada del proceso. Posteriormente recorre todas las acciones del MDCS (`allActions`) aplicando cada una de las reglas de transformación (de acciones MDCS a actividades BPEL) anteriormente expuestas y finalmente genera la respuesta (`psm_bpel::Reply`) del proceso.

Mayor detalle de la implementación de la regla se encuentra en el Apéndice 3.

```
/* Creando las actividades del proceso BPEL - Primero el sequence general
y luego el resto -------------*/
initSeq := object psm_bpel::Sequence {

/* Recibe la solicitud inicial del cliente ------------------------------------ */
                activity    += object psm_bpel::Receive {
                partnerlink := 'client';
                operation   := operation.name;
                portType    := operation.name+'PT';
                variable       := 'input';
                receiveVar     := object psm_bpel::VarOpWS {
                        name := operation.name+'Request';
                        part += allParameters->collect(p|makePartParameter(p))
                        };
                createInstance := 'yes';
                };

/* El cuerpo del proceso BPEL ------------------------------------------------- */
                activity += allActions->collect(a|a.toActivity())
                        ->reject(a|a.oclIsTypeOf(psm_bpel::Activity));

/* Respuesta del proceso BPEL ------------------------------------------------- */
                activity += object psm_bpel::Reply {
                        partnerlink    := 'client';
                        operation      := operation.name;
                        portType       := operation.name+'PT';
                        variable       := 'output';
                        replyVar       := object psm_bpel::VarOpWS {
                                name := operation.name+'Response';
                                part += object psm_bpel::Part {
                                name :=operation.mdcsOpDef.returnValue
                        }
                };
        }
```

Figura 7.45 Extracto de la regla de transformación que genera el cuerpo del proceso BPEL

7.2.3.4 Reglas de transformación PSM-A-PSM

El PSM-BPEL que se ha obtenido en las transformaciones anteriores carece de las variables necesarias para los mensajes de entrada/salida para la invocación de los servicios componente. Por lo que se requiere una transformación adicional al modelo PSM antes de generar el código final.

La transformación PSM-A-PSM para la generación de variables faltantes es la siguiente (ver Figura 7.46):

- **addVariablesFaltantes**: esta transformación consulta todas las actividades del PSM-BPEL obtenido en la transformación anterior y genera todas las variables de entrada/salida necesarias para la invocación de las operaciones de los servicios componente. Las variables obtenidas se agregan a las variables locales ya existentes.

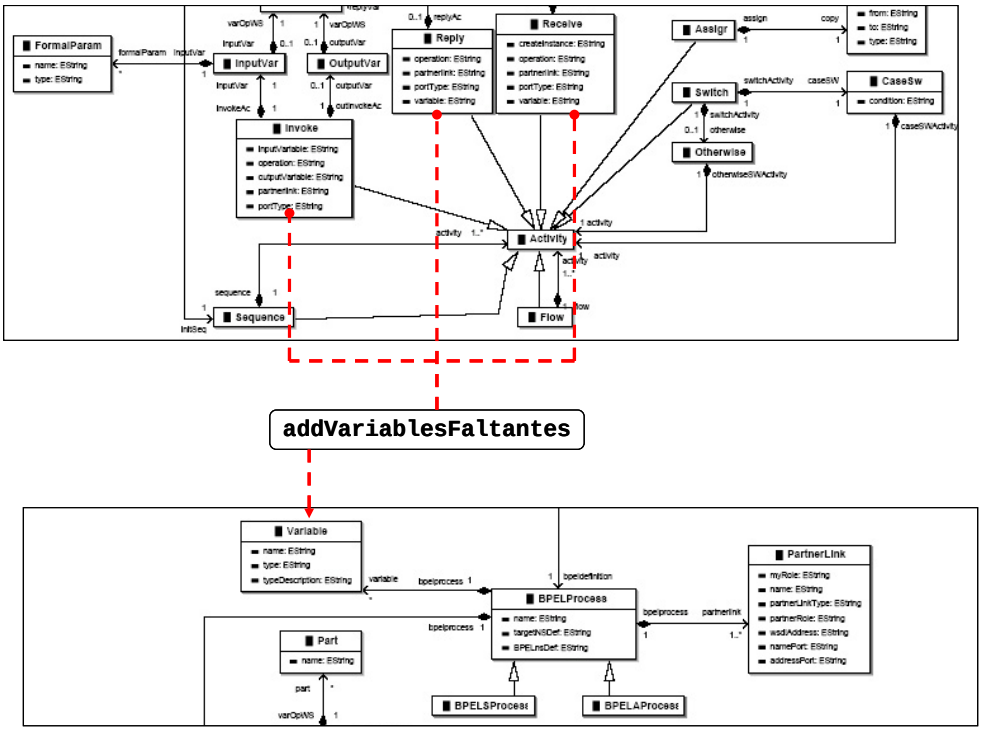


Figura 7.46 Agregación de variables faltantes al PSM-BPEL

Después de aplicar esta regla de transformación, el PSM-BPEL posee las variables faltantes `BNQuoteServicegetPriceRequest/Response` y `AmazonasinSearchRequest/Response` tal como lo muestra la Figura 7.47.

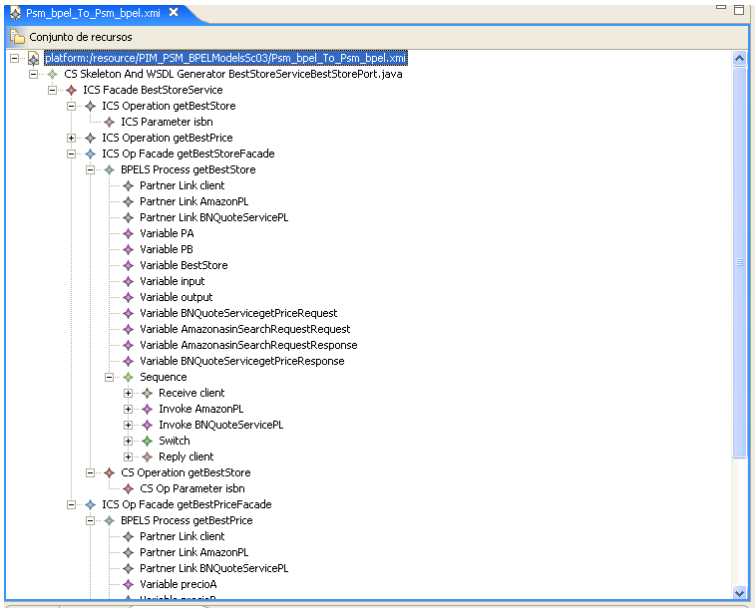


Figura 7.47 PSM-BPEL agregando variables faltantes

La Figura 7.48 muestra como ejemplo de la regla expuesta la implementación en QVT-operacional de Together de su núcleo.

```
/* Una vez generado el proceso BPEL inicial, se da una segunda pasada
para agregar las variables adicionales -----*/

mapping addVariablesFaltantes(inout bpelmodel:psm_bpel::BPELProcess)
:psm_bpel::BPELSProcess
{
        init {
                var allActivities  := bpelmodel.initSeq.activity;
                var allActivitiesExceptFirst         := allActivities
                ->excluding(allActivities->first())->asOrderedSet();
                var allActivitiesExceptFirstAndLast  := allActivitiesExceptFirst
                ->excluding(allActivitiesExceptFirst->last())->asOrderedSet();

                var allVariables   := getVariables(allActivitiesExceptFirstAndLast);
                var allVariablesWithOutRepeated := eliminateRepeated(allVariables)
                        ->asOrderedSet();

                bpelmodel.variable += allVariablesWithOutRepeated;
                result := bpelmodel.oclAsType(psm_bpel::BPELSProcess);
        }
}
```

Figura 7.48 Núcleo de la regla de transformación `addVariablesFaltantes`

El algoritmo se basa en un doble recorrido del modelo PSM obtenido en la transformación anterior para detectar las actividades de invocación de operaciones en servicios componente para luego agregar en las variables locales del proceso BPEL las variables para los mensajes de entrada/salida necesarios y faltantes.

Más detalles de la transformación se encuentran en el Apéndice 3.

7.2.3.5 Reglas de transformación PSM-A-Código

Una vez que se ha obtenido el PSM con la definición completa de variables, un conjunto de reglas de transformación *Modelo-A-Texto* permiten generar los artefactos necesarios para las plataformas Java-Axis y *Oracle BPEL Process*.

Dado que los algoritmos para generación de fachadas son semejantes a los ya presentados en los escenarios 2 y 3, se omite su explicación en este apartado. La generación del proceso BPEL está dado por los siguientes pasos de transformación, desde el modelo PSM-BPEL hacia código BPEL. Se ofrece cada sección del algoritmo en lenguaje natural, una muestra de su implementación en *MOFScript* y un ejemplo del código generado. El código completo se encuentra en el Apéndice 3 :

El **primer paso** genera el *tope del proceso*:

1. Generar el inicio del proceso.
    a. Inicializa el nombre del proceso a partir `BPELProcess.name`
    b. Inicializa el espacio de nombres destino a partir de `BPELProcess.targetNSDef`
    c. Inicializa el espacio de nombre del proceso a partir de `BPELProcess.BPELnsDef`
    d. Inicializar los espacios de nombre restantes (`xsd`, `bpws`) a partir de valores constantes.
    e. Para cada *partnerlink*:
        i. Genera su espacio de nombres a partir de su rol (`partnerRole`) y la dirección de su puerto (`addressPort`)

La Figura 7.49 muestra el código de implementación del algoritmo anterior:

```
// Genera el elemento <process...> del Proceso BPEL ----------------------
  psm_bpel.BPELSProcess::generateProcessBegin()
  {
    '<process name="' self.name '"' newline(1)
    tab(2) 'targetNamespace="' self.targetNSDef '"' newline(1)
    tab(2) 'xmlns:tns="' self.targetNSDef '"' newline(1)
    tab(2) 'xmlns="' self.BPELnsDef '"' newline(1)
    tab(2) 'xmlns:xsd="http://www.w3.org/2001/XMLSchema"' newline(1)
    tab(2) 'xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"'
        newline(1)
    self.generatexlmnsPartnerLinks() '>' newline(1)
  }
```

Figura 7.49 Generación del tope del proceso BPEL

La Figura 7.50 muestra un ejemplo del código generado usando este algoritmo:

```
<process name="getBestStore"
targetNamespace="urn:BestStoreServicegetBestStore"
xmlns:tns="urn:BestStoreServicegetBestStore"
xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:AmazonService="http://levitico:8080/AmazonWebModule/services/AmazonWSPort"
xmlns:BNQuoteServiceService="http://levitico:8080/BNWebModule/services/BNWSPort"
>
```

Figura 7.50 Paso 1: ejemplo de código de inicio de proceso generado

El **segundo paso** genera los *enlaces a los servicios componente*:

2. Generar la sección de *PartnerLinks*.
    a. Genera el inicio de la sección *PartnerLink*.
    b. Para cada `BPELProcess.partnerlink`:
        i. Genera el enlace al servicio componente a partir de su tipo (`partnerLinkType`) y roles (`myRole` y `partnerRole`).
    c. Genera el fin de la sección *PartnerLink*.

La Figura 7.51 muestra la implementación del algoritmo anterior en *MOFScript*:

```
// Genera cada PartnerLink ------------------------------------------------------------
psm_bpel.PartnerLink::generatePartnerLInk()
{
        tab(1) '<partnerLink name="' self.name '"' newline(1)
        if (self.name.endsWith('client'))
        {
                tab(2) 'partnerLinkType="tns:' self.partnerLinkType '"' newline(1)
        }
        else
        {
                tab(2) 'partnerLinkType="' self.partnerRole ':'self.partnerLinkType '"'
                newline(1)
        }
        if (self.myRole.trim().size() <> 0)
        {
                tab(2) 'myRole="' self.myRole '"' newline(1)
        }
        if (self.partnerRole.trim().size() <> 0)
        {
                tab(2) 'partnerRole="' self.partnerRole '"' newline(1)
        }
        tab(1) "/>" newline(1)
}
```

Figura 7.51 Generación de los enlaces a servicios componente

La Figura 7.52 muestra un ejemplo del código generado usando este algoritmo:

```
<partnerLinks>
<partnerLink name="client"
partnerLinkType="tns:getBestStore_clientLT"
myRole="getBestStoreService"
/>
<partnerLink name="AmazonPL"
partnerLinkType="AmazonService:AmazonLT"
partnerRole="AmazonService"
/>
<partnerLink name="BNQuoteServicePL"
partnerLinkType="BNQuoteServiceService:BNQuoteSe
rviceLT"
partnerRole="BNQuoteServiceService"
/>
</partnerLinks>
```

Figura 7.52 Paso 2: código generado para los enlaces a los servicios componente

El **tercer paso** genera la *sección de variables* utilizadas por el proceso:

3. Generar la sección de variables.

a. Genera el inicio de la sección de variables.
b. Para cada variable (`BPELProcess.Variable`):
    i. Genera la variable del proceso a partir de su tipo (`type`) y descripción (`description`).
c. Genera el fin de la sección de variables.

Un extracto del código que implementa el algoritmo anterior está dado en la Figura 7.53.

```
// Genera 1 variable ------------------------------------------------------
  psm_bpel.Variable::generateVariable()
  {
    if (self.name.equals("input") or self.name.equals("output"))
    {
    tab(1) '<variable name="' self.name '"' newline(1)
    tab(2) self.type '="tns:' self.typeDescription '"/>' newline(1)
    }
    else
    {
    tab(1) '<variable name="' self.name '"' newline(1)
    tab(2) self.type '="' self.typeDescription '"/>' newline(1)
    }

  }
```

Figura 7.53 Generación de las variables del proceso

La Figura 7.54 muestra un ejemplo de generación de variables utilizando este algoritmo.

```
<variables>
<variable name="PA"
type="xsd:float"/>
<variable name="PB"
type="xsd:float"/>
<variable name="BestStore"
type="xsd:string"/>
<variable name="input"
messageType="tns:getBestStoreRequestMessage"/>
<variable name="output"
messageType="tns:getBestStoreResponseMessage"/>
</variables>
```

Figura 7.54 Paso 3 : ejemplo de variables generadas

El **cuarto paso** genera el *cuerpo del proceso*:

4. Generar el cuerpo del proceso.
    a. Genera el inicio del cuerpo del proceso.

b. General el detalle del cuerpo del proceso.
    i. Para cada actividad del proceso BPEL:
        1. Si la actividad es *Invoke*:
            a. Copia los parámetros a la variable de entrada del *Invoke*.
            b. Genera el *Invoke* con sus variables de entrada y/o salida (definiendo su partes) y *partnerLinks*.
            c. Copia el resultado en la variable de salida del *Invoke*.
        2. Si la actividad es *Reply*:
            a. Copia los valores a la variable de salida.
            b. Genera el *Reply* con la variable de salida.
        3. Si la actividad es *Receive*:
            a. Genera el *Receive* para el cliente si el enlace (`partnerLink`) es el cliente, en caso contrario genera el *Receive* general.
        4. Si la actividad es *Assign*:
            a. Genera la asignación para `variable` si el tipo de la expresión a copiar es *variable*, en caso contrario genera la asignación para `expresión`.
        5. Si la actividad es *Switch*:
            a. Genera la actividad para la condición verdadera.
            b. Genera la actividad para la condición falsa.
        6. Si la actividad es *Sequence*:

a. Para cada actividad incluida en la secuencia: aplica los pasos 5 al 6 anteriores recursivamente.

c. Genera el fin del cuerpo del proceso.

5. Generar el fin del proceso.

La Figura 7.55 muestra la implementación central del algoritmo anterior. El detalle de programación para la generación del código de cada actividad BPEL se omite por razones de espacio pero se pueden consultar mayores detalles en el Apéndice 3.

```
psm_bpel.BPELSProcess::generateBodySection()
  {
    generateBodySectionBegin()
    self.initSeq.activity->forEach(a) { a.generateActivity() }
    generateBodySectionEnd()
  }

// Genera el inicio del cuerpo del proceso ------------------------
--------------------
  module::generateBodySectionBegin()
  {
    '<sequence>' newline(1)
  }

// Genera el código para una actividad BPEL -----------------------
----------------
psm_bpel.Activity::generateActivity()
{
 if (self.oclIsKindOf(psm_bpel.Invoke))
 self.generateInvoke()
 else
 if (self.oclIsKindOf(psm_bpel.Reply))
 self.generateReply()
 else
 if (self.oclIsKindOf(psm_bpel.Receive))
 self.generateReceive()
 else
 if (self.oclIsKindOf(psm_bpel.Assign))
 self.generateAssign()
 else
 if (self.oclIsKindOf(psm_bpel.Switch))
 self.generateSwitch()
 else
 if (self.oclIsKindOf(psm_bpel.Sequence))
 self.generateSequence()
}
// Genera el fin del cuerpo del proceso ---------------------------
----------------
  module::generateBodySectionEnd()
  {
    '</sequence>' newline(1)
  }
}
```

Figura 7.55 Algoritmo central para la generación del proceso BPEL

La Figura 7.56 muestra un ejemplo de código generado para un proceso BPEL mediante el algoritmo anterior.

```
<sequence>
<receive partnerLink="client"
portType="tns:getBestStorePT"
operation="getBestStore"
variable="input"
createInstance="yes" />
<assign>
<copy>
<from variable="input" part="payload"
query="/tns:getBestStoreRequest/tns:isbn"></from>
<to variable="AmazonasinSearchRequestRequest" part="isbn"/>
</copy>
</assign>
<invoke partnerLink="AmazonPL"
portType="AmazonService:AmazonJavaWS"
operation="asinSearchRequest"
inputVariable="AmazonasinSearchRequestRequest"
outputVariable="AmazonasinSearchRequestResponse" />
<assign>
<copy>
<from variable="AmazonasinSearchRequestResponse" part=""></from>
<to variable="PA"/>
</copy>
</assign>
<reply partnerLink="client"
portType="tns:getBestStorePT"
operation="getBestStore"
variable="output" />
</sequence>
```

Figura 7.56 Paso 4: ejemplo de código BPEL generado

Las transformaciones *Modelo-A-Texto* para la generación de los artefactos restantes (la Interfaz WSDL de la Operación Componente, la Interfaz *Wrapper* a los servicios componente, el descriptor de procesos, el descriptor de proyectos y el constructor de proyectos) se muestran en el Apéndice 3, así como un ejemplo de código generado para los mismos.

7.2.4 Escenario 4: Especialización de Servicios

La especialización de servicios se implementa haciendo uso del patrón de diseño *Decorador*[60]. Considerando que no existen mecanismos que de forma nativa permitan la especialización de servicios, se optó por este patrón que permite su implementación a partir de la agregación. Así, la idea fundamental que se propone es que la especialización de un servicio base (propio o ajeno) en otro derivado (propio) se logra mediante la conversión de la especialización en una agregación (composición) de servicios. Entonces, una vez que se obtiene esta última, se puede reuti-

lizar la estrategia de generación de código definida en el escenario 3 para la generación de código final.

En este contexto, la generación de código para la especialización se lleva a cabo en dos etapas principales: **(1)** una transformación de la especialización de servicios en agregación de servicios y **(2)** un proceso de generación de código para la composición reutilizando los pasos definidos para el escenario 3. En esta sección la explicación sólo enfatiza la primera etapa.

La **transformación del servicio especializado en una agregación de servicios** se lleva a cabo en cuatro pasos (Figura 7.57):

(1) En el **primer paso** se copia el servicio derivado como servicio compuesto; después, se agrega el servicio base como servicio componente del servicio compuesto.

(2) En el **segundo paso** se generan las operaciones especializadas en el servicio compuesto acorde a los casos de especialización de servicios que fueron discutidos en el capítulo 4 (sección 4.2.2.2): refinamiento de la signatura de la operación, refinamiento de la lógica de la composición de la operación y agregado de una nueva operación.

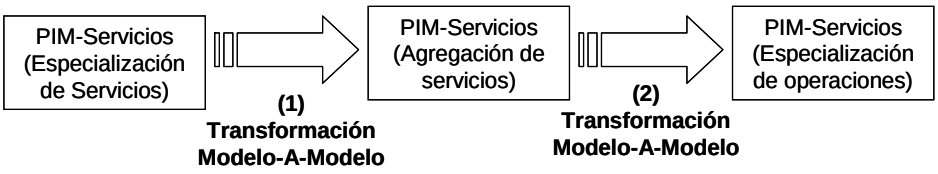


Figura 7.57 Transformación de una especialización de servicios en agregación de servicios

(3)-**(4)** El **tercer** y **cuarto paso** son los mismos que se han aplicado para el escenario 3: obtención de un modelo PSM-BPEL con definición completa de variables y obtención de las clases fachada y proceso BPEL, tal como lo ilustra la Figura 7.58.

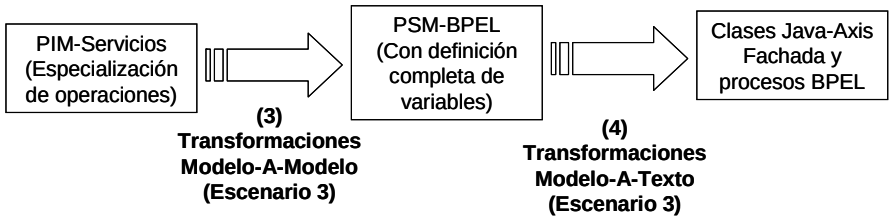


Figura 7.58 Transformación de la agregación de servicios al código Java-Axis y BPEL

7.2.4.1 Metamodelo PIM

La Figura 7.59 muestra las clases que el PIM de Servicios incluye para el manejo de especialización de servicios.

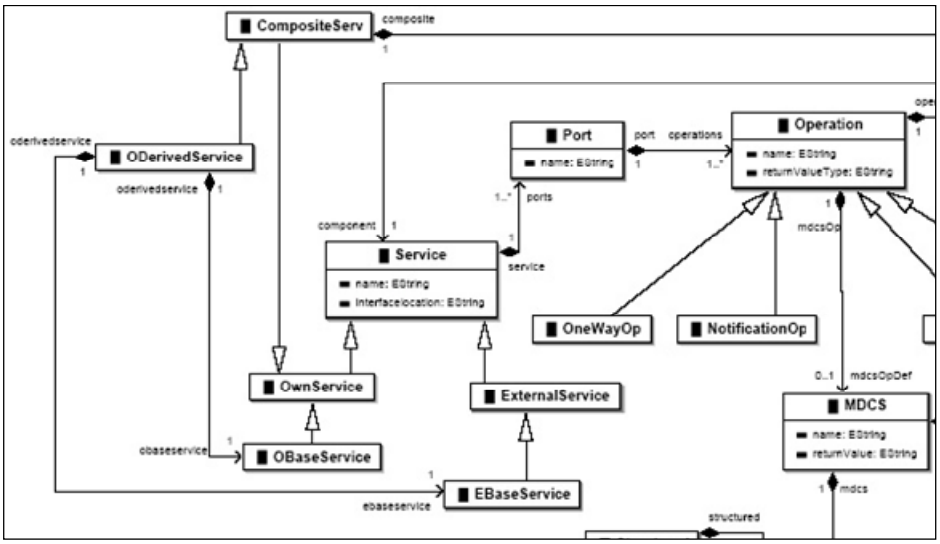


Figura 7.59 Clases del PIM de Servicios para especialización

Todas son especializaciones de las clases de servicios ya existentes por lo que heredan sus atributos y relaciones:

- **ODerivedService**: dado que el servicio derivado será manejado después de la primer transformación de modelos como servicio compuesto (*CompositeServ*), se ha incluido como una especialización del mismo. Como los servicios compuestos son servicios propios, los servicios derivados son también servicios propios.

- **OBaseService**: un servicio propio que se desea especializar en un servicio propio.
- **EBaseService**: un servicio ajeno que se desea especializar en un servicio propio.

Como ejemplo de especialización de servicios, la Figura 7.60 muestra el PIM para el servicio `TravelService` especializado como `TravelHotelService` expuesto en el capítulo 4.

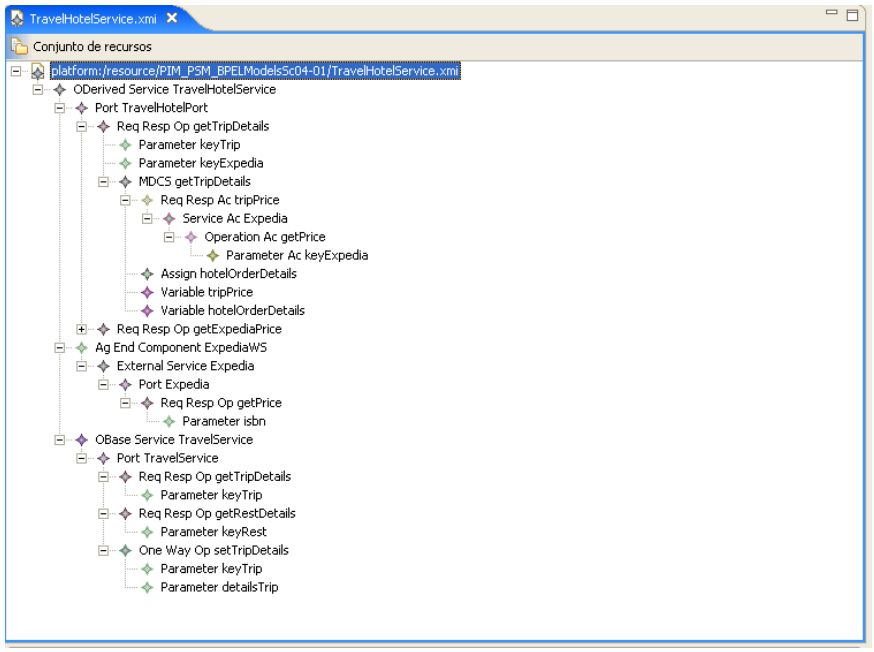


Figura 7.60 Servicio especializado `TravelService`

7.2.4.2 Metamodelo PSM

Las plataformas tecnológicas son las mismas que para el escenario 3 (Java, Axis y BPEL) por lo que se omite su explicación. Refiérase a los metamodelos PSM del mencionado escenario para su explicación.

7.2.4.3 Reglas de transformación PIM-A-PIM

El siguiente conjunto de reglas de transformación *Modelo-A-Modelo* se han definido para los pasos (1) y (2) de generación de servicio compuesto a partir del servicio especializado (ver Figuras 7.57 y 7.61):

- **makeComposite**: a partir del servicio Derivado (*ODerivedService*) se genera el servicio compuesto equivalente (*CompositeServ*): el nombre del servicio (`name`) se mantiene, así como sus puertos (`ports`) y los posibles servicios componentes que pudiera tener (`agEndComponents`). El servicio base (*OBaseService* o *EBase-Service*) se agrega como un componente más en el servicio compuesto.
- **convertToOwnService**: convierte el servicio base (propio o ajeno) en el servicio componente que se agregará en el servicio compuesto.
- **makePort**: crea un puerto (*Port*) en el servicio compuesto. Este puerto contendrá además de las operaciones especializadas, las no especializadas y las que han sido agregadas en el servicio derivado.
- **toNewMDCS<Action>Op:** para aquellas operaciones cuya signatura ha sido refinada así como también la lógica de su MDCS (agregando nuevas acciones). Esta regla transforma el MDCS en uno nuevo que llama a la acción original y agrega otras nuevas.
- **toOneActionActionMDCSfor<Action>Op**: se utiliza para generar las operaciones no especializadas del servicio base en el servicio compuesto. Permite generar una operación con un MDCS que posee una acción la cual invoca a la operación original del servicio base (que para el servicio compuesto será un servicio componente).

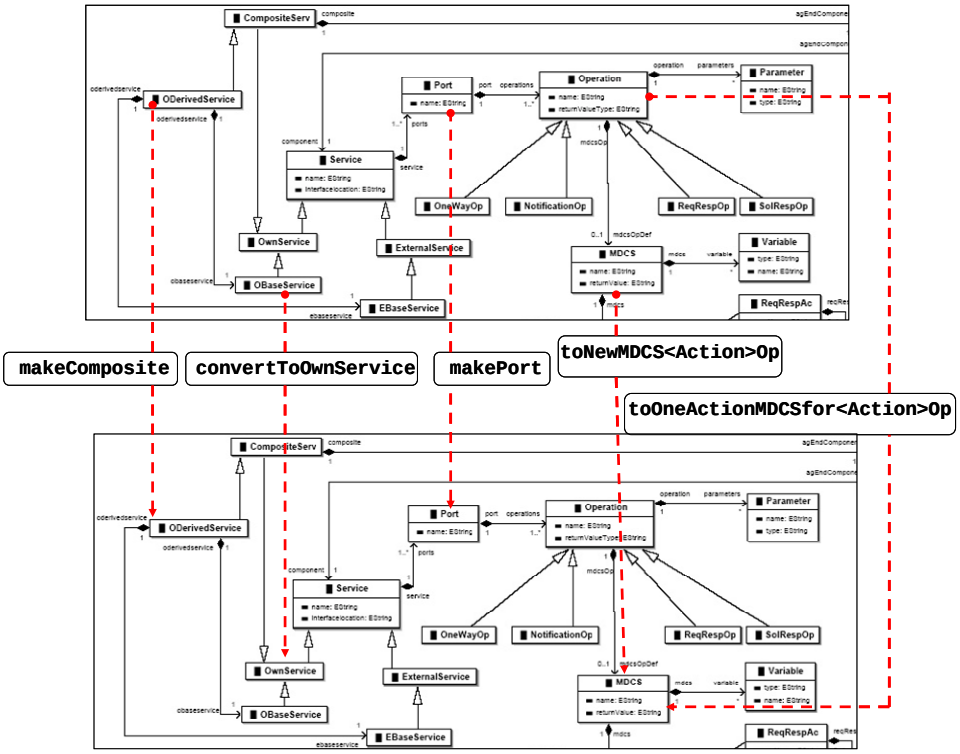


Figura 7.61 Reglas de transformación para convertir un servicio especializado en una agregación de servicios

La Figura 7.62 muestra el código de la transformación `makeComposite`.

```
/* Convierte un servicio derivado en otro servicio compuesto equivalente ---------------------------- */
mapping makeComposite(in model: kernel::ODerivedService) : kernel::CompositeServ
{
        init
        {
        }
        object {
                name          := model.name;
                ports         += model.ports->collect(p|makePort(p))->asOrderedSet();
                agEndComponents := model.agEndComponents;
                agEndComponents += object kernel::AgEndComponent {
                        name      := '_BS_';
                        component := convertToOwnService(model.obaseservice)
                };
        }
}
```

Figura 7.62 Implementación de la regla de transformación `makeComposite`

La regla recibe en su entrada el servicio derivado (`kernel::ODerivedService`) y obtiene el servicio compuesto equivalente (`ker-`

`nel::CompositeServ`). En este servicio compuesto se generan a partir del servicio derivado su nombre (`name`), puertos (`model.ports`) y se agrega el servicio base como servicio componente mediante la operación `convertToOwnService`.

La Figura 7.63 muestra el PIM de especialización del servicio `TravelService` transformado en agregación de servicios. El ejemplo completo se explica en el Apéndice 4.

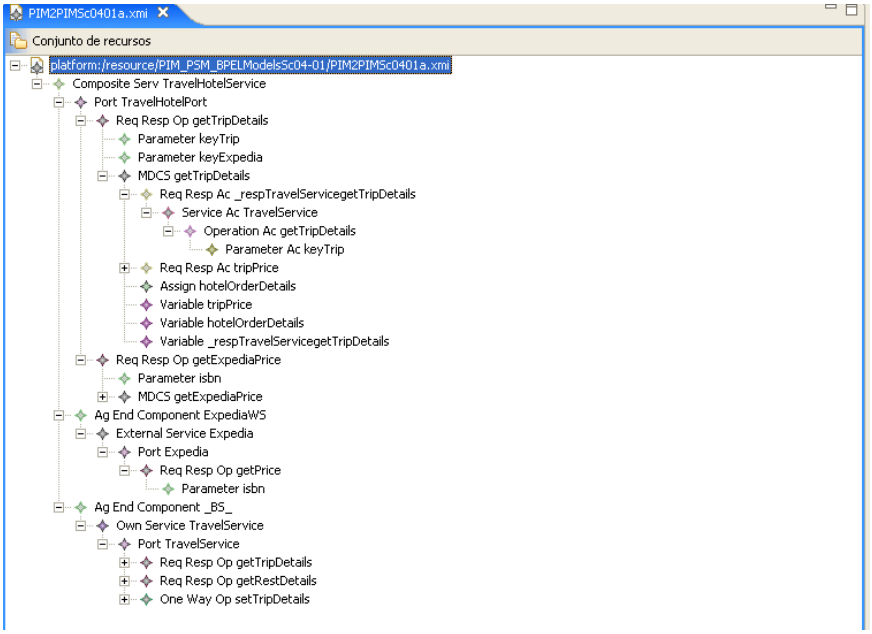


Figura 7.63 Servicio especializado transformado en agregación

7.2.4.4 Reglas de transformación PIM-A-PSM y PSM-A-Código

Con el PIM de agregación de servicios que se ha obtenido a partir de las transformaciones anteriores, se aplican las reglas de transformación PIM-A-PSM del escenario 3 para obtener el PSM-BPEL con definición completa de variables. Una vez que se obtiene éste, la generación de código final es llevada a cabo con las mismas reglas de transformación *Modelo-A-Texto* para el mencionado escenario.

7.3 Conclusiones

Se ha presentado, basado en el marco de trabajo de MDA, la estrategia de generación de código de los componentes que implementan los aspectos de integración de las aplicaciones Web especificadas en OOWS. Esta estrategia representa la implementación de las correspondencias entre los modelos y sus representaciones software, del paso de Modelado de Integración establecido en la redefinición del enfoque metodológico del capítulo 5.

Las estrategias presentadas generan ahora los componentes software que permitirán a la aplicación Web: (1) el consumo de funcionalidad externa a través de servicios Web; (2) la producción de funcionalidad de la capa de aplicación a través de servicios Web; (3) la producción de funcionalidad compuesta a partir de la agregación y especialización de servicios Web propios y ajenos y (4) la integración de los servicios Web con los componentes software internos.

Los componentes software generados forman la subcapa de Lógica de Integración, destinada al consumo y producción de servicios Web de la aplicación Web. Las fachadas generadas representan la interfaz interna a la subcapa y facilitan el acoplamiento de los componentes de la capa de Presentación. También, las interfaces WSDL generadas para los servicios propios, ofrecen la API de la aplicación Web que permite ahora que aplicaciones externas puedan consumir su funcionalidad y lograr con ello la integración completa.

Capítulo 8
Conclusiones

Este capítulo final ofrece un resumen del conjunto de contribuciones finales del presente trabajo de investigación, una breve discusión con posibles cuestionamientos sobre las decisiones tomadas, las publicaciones que se han derivado, así como los trabajos de investigación futuros que se pueden llevar a cabo a partir de lo realizado en esta tesis.

8.1 Contribuciones

Todas las contribuciones de esta tesis están enmarcadas en el contexto del método OOWS. Todas han sido dirigidas para aportar a la extensión del método para su habilitación en la especificación de aplicaciones Web que integran servicios Web:

- Se ha realizado una revisión del estado del arte en las disciplinas de Ingeniería Web y Procesos Negocio-a-Negocio considerando el papel que juegan los servicios Web y las aplicaciones Web en las mismas. Se han detectado áreas de oportunidad que permiten su acercamiento complementario y que han sido la base del presente trabajo de tesis. Este acercamiento se ha ofrecido en el contexto de una extensión al método OOWS.
- Se ha agregado a la fase de Especificación del Sistema del método OOWS una etapa más de modelado conceptual: el Modelado de Integración. Con esta etapa se logra especificar los aspectos de integración de la aplicación Web mediante dos modelos: de Servicios y Dinámico de Composición de Servicios. Esta nueva etapa se agrega

sin modificar las etapas anteriores del método, con algunas adecuaciones a sus modelos y técnicas.

- El Modelo de Servicios es también otra contribución. Mediante este modelo es posible especificar la funcionalidad que produce y consume la aplicación. Una contribución novedosa de esta tesis es la inclusión de relaciones de agregación y especialización entre servicios enriquecidas mediante un marco de trabajo multidimensional como un mecanismo que permite la definición conceptual de funcionalidad de la aplicación. Estas relaciones permiten la definición de servicios compuestos con mayor precisión que la forma común de especificación mediante sólo modelos dinámicos.
- Otra contribución de esta tesis es el Modelo Dinámico de Composición de Servicios complemento al Modelo de Servicios. Este modelo es necesario para la definición de operaciones en servicios compuestos y se apoya en las relaciones estructurales del Modelo de Servicios.
- Dentro de la fase de modelado conceptual del método OOWS se han redefinido las primitivas del Mapa Navegacional. Buscando uniformidad para la consulta y actualización de datos locales (proveniente del Diagrama de clases), así como en el consumo de funcionalidad propia y ajena (proveniente del Modelo de Servicios) se ha ofrecido una actualización de la primitiva Clase navegacional por Unidad de Interacción. Esta unidad ofrece compatibilidad con la Clase navegacional enriqueciéndose para soportar el uso de los servicios Web a través del Mapa Navegacional. Una contribución importante de esta tesis y que diferencia a las soluciones existentes en otros métodos de Ingeniería Web es que se ha respetado el principio de separación de aspectos en todos estos modelos. Todos los modelos ofrecidos capturan aspectos diferenciados, pero a su vez se integran para la especificación de la aplicación. Las unidades de interacción se definen a través de metamodelos MOF.
- El Modelo de Presentación también se ha extendido para incluir patrones de presentación para las Unidades Funcionales. Los patrones de presentación para las Unidades de Información mantienen com-

patibilidad con los patrones de presentación ya existentes para las Clases navegacionales originales.

- Se ha definido también un método para la obtención de Mapas Navegacionales a partir de un Modelo de proceso Negocio-a-Negocio. La estrategia define modelos y técnicas de transformación que permiten la obtención de una aplicación Web prototipo que puede luego ser revisada y adecuada por el diseñador.
- Finalmente, aunque no de menor importancia, se ha ofrecido un conjunto de estrategias de generación de código a partir de los nuevos modelos introducidos. Estas estrategias han sido diseñadas de acuerdo al marco de trabajo MDA.

8.2 Algunos cuestionamientos sobre las decisiones tomadas

Las decisiones tomadas fueron resultado de diversos cuestionamientos surgidos a lo largo del trabajo de investigación. La respuesta a ellos se ve reflejado en el resultado brindado en esta tesis. Algunas otras pueden surgir también, aquí se intenta ofrecer una breve discusión al respecto:

- *La tesis se ha limitado al método OOWS, ¿Qué contribuciones podría brindar a otros métodos?* A partir de la revisión del estado del arte, se detectaron áreas de oportunidad en las que pueden brindarse contribuciones. De manera general, algunas de ellas podrían ser: la definición de modelos particulares para la especificación de servicios Web, su integración con los modelos restantes y el mantenimiento del principio de separación de aspectos en la adecuación y/o extensión de los métodos.
- *¿Cuál otra variante existe para la propuesta ofrecida de adecuación al método OOWS?* En la fase de Desarrollo de la Solución, en la definición de la arquitectura software, se puede incluir la subcapa de lógica de integración como parte de la Capa de Presentación. Esto es otra opción posible, ya que si consideramos que la responsabilidad de dicha capa es ofrecer la interfaz de la aplicación Web, esta res-

ponsabilidad podría ampliarse al ofrecer comunicación, tanto a actores humanos y no humanos (como API).

- *En la propuesta de esta tesis el principal cliente del Modelo de Servicios es el Mapa Navegacional, ¿Qué otra opción podría brindarse?* Así como el Mapa Navegacional captura los requisitos de la Interfaz de Usuario (IU), el conjunto de servicios propios del Modelo de Servicios definen su API. Además de las opciones de construcción de servicios propios ofrecidos en la tesis, otra posible opción podría ser a partir del conocimiento capturado en los modelos restantes, por ejemplo el mismo Modelo Navegacional. Así, la API se podría definir, no sólo a partir de la funcionalidad de la capa de Aplicación, sino a partir de cualquier otra presente en la aplicación Web.
- *¿Porqué el método de obtención del Mapa Navegacional a partir del Modelo de proceso Negocio-a-Negocio es semiautomático?* El enfoque de la tesis esta orientado, principalmente, a los aspectos de integración de datos y funcionalidad en la aplicación Web, a partir de servicios Web. Esto forma parte de una solución completa que debe considerar también los procesos Negocio-a-Negocio. Por eso se abordó este aspecto también, pero limitado a ofrecer un punto inicial de solución que pueda ser retomado en otros trabajos de investigación con mayor amplitud y profundidad, donde la automatización completa pueda ser posible.

8.3 Trabajo futuro

El trabajo futuro que se puede llevar a cabo a partir del trabajo de esta tesis puede incluir lo siguiente:

- Aplicaciones Web que incluyen servicios con operaciones asíncronas. El uso de este tipo de operaciones está resultando cada vez más necesario para los nuevos Modelos de negocio y sería una continuación lógica del trabajo realizado en esta tesis. Esto implicaría: un estudio e inclusión de operaciones asíncronas en el Modelo de Servicios y Dinámico de Composición de Servicios. Estudio y posible de-

finición (o redefinición) de las Unidades de Interacción para la especificación de aplicaciones Web. En particular, esto implicaría un estudio detallado del papel que juegan las operaciones *Request-response* y *Solicite-response* en su modalidad asíncronica.

- La solución que se ha ofrecido en la obtención del Mapa Navegacional a partir del Modelo de procesos Negocio-a-Negocio es semi-automática y requiere revisión por parte del diseñador del Mapa Navegacional, por lo cual otro trabajo podría ser lograr la automatización completa del proceso de obtención de aplicaciones Web a partir de modelos de proceso distribuidos.
- En este mismo contexto, la solución que se ha ofrecido sigue un enfoque *Top-Down*, en el cual se buscó trazabilidad desde el proceso de negocio hasta la aplicación Web; sin embargo un par de enfoques adicionales es también posible: (1) *Bottom-up*: partiendo de procesos Negocio-a-Negocio ya existentes y buscando su incorporación a la aplicación Web por construir o (2) *Intermedia*: partiendo de procesos Negocio-a-Negocio y aplicación Web existentes, buscando la extensión de está última para su incorporación en el primero. Esto puede dar origen a dos trabajos adicionales.
- La implementación de la selección dinámica de servicios; que requiere de plataformas tecnológicas que ofrezcan mecanismos específicos para este tipo de tarea, los cuales no ofrecen plataformas tecnológicas como BPEL.
- La implementación del conjunto de correspondencias desde el Modelo de Presentación extendido (expuesto en el capítulo 5) hacia componentes software de la capa de presentación.
- La implementación del proceso de importación *Texto-a-Modelo* del escenario 1 para servicios Web, basados en REST.
- Considerando el papel cada vez mayor que está jugando la Web social (también conocida como la Web 2.0) otro trabajo podría incluir la extensión del método OOWS con primitivas que permitan la especificación de este tipo de aplicaciones anexado a las primitivas de consulta y actualización de datos, así como de producción y consumo de funcionalidad que se han definido. Un área interesante a ex-

plorar podría ser la forma en la que todos estos contenidos que se están generando por los usuarios (más que por los diseñadores Web) puedan ser exportados e importados mediante servicios y con ello lograr su incorporación en nuevas y más ricas aplicaciones Web.

8.4 Publicaciones relacionadas con la tesis

Las publicaciones [86, 87, 88, 89, 90, 91, 92, 93, 94] que se han derivado a partir de este trabajo de investigación son las siguientes:

- **Quintero R.**, Pelechano V., Fons J., Pastor O. *Desarrollo de Aplicaciones Web mediante la aplicación de MDA a OOWS.* XII Congreso Internacional de Computación (CIC 2003). Instituto Politécnico Nacional. México, D.F. 13-17 octubre de 2003.
- **Quintero R.**, Pelechano V., Fons J., Pastor O. *Desarrollo de Sistemas Basados en Web mediante la aplicación de MDA a OOWS*. 10mo. Congreso Internacional de Investigación en Ciencias Computacionales (CIIC 2003). Oaxtepec, Morelos, México. 22-24 octubre de 2003.
- **Quintero R.**, Pelechano V., Pastor O., Fons J. *Aplicación de MDA al Desarrollo de Aplicaciones Web en OOWS*. Page(s): 379-388. Jornadas de Ingeniería de Software y Base de Datos (JISBD), VIII-2003 – November, Alicante(Spain). Ernesto Pimentel, Nieves Brisaboa, Jaime Gómez, 84-668-3836-5, 2003.
- **Quintero R.**, Pelechano V., Torres V., Ruiz M. *Una solución a la integración de los métodos de ingeniería Web y las aplicaciones B2B. Un caso de estudio*. Page(s): 301 - 312, Workshop Iberoamericano de Ingenieria de Requisitos y Ambientes Software (IDEAS), Is International, VIII, 2005 - May, Valparaiso (Chile), Hernán Astudillo, 2005.
- Torres V., **Quintero R.**, Ruiz M., Pelechano V. *Towards the Integration of Data and Functionality in Web Applications.* A Model Driven Approach, Page(s): 33 - 38, Conference on Advanced Information Systems Engineering Forum (CAiSE Forum), Is Interna-

tional, 17, 2005 - June, Porto (Portugal), Orlando Belo, Johann Eder, Oscar Pastor, João Falcão e Cunha , 972-752-078-2, 2005

- **Quintero R.**, Torres V., Ruiz M., Pelechano V. *Modelado de aplicaciones web colaborativas basadas en servicios y dirigidas por procesos de negocio B2B*. Page(s): 123 - 132, Jornadas Científico-Técnicas en Servicios Web (JSWEB), I, 2005 - September, Granada (Spain), Esperanza Marcos, José Alonso, Valeria de Castro, José Carlos del Arco, 84-9732-455-2, 2005
- **Quintero R.**, Torres V., Ruiz M., Pelechano V. *A Conceptual Modeling Approach for the Design of Web Applications based on Services*. ACMSE 2006. ISBN: 1-59593-315-8. Page(s): 464-469. Melbourne, Florida. USA.
- **Quintero R.**, Pelechano V. *Conceptual Modeling of Service Composition using Aggregation and Specialization Relationships*. ACMSE 2006. ISBN: 1-59593-315-8. Page(s): 452-457. Melbourne, Florida. USA.
- **Quintero R.**, Torres V., Pelechano V. *Model Centric Approach of Web Services Composition*. WEWST/ECOWS 06 (European Conference on Web Services). Zurich, Switzerland. 2006. Book Chapter. Book Series: Whistein Series in Software Agent Technologies and Autonomic Computing. Book: Emerging Web Services Technology. ISBN: 978-3-7643-8447-0 (Print) 978-3-7643-8448-7 (Online). Page(s): 65-81. Editor: Birkhäuser Basel. Springer.

APÉNDICE 1

Ejemplo de Importación de Servicios Ajenos

A continuación se muestra un ejemplo de implementación del escenario 1 (*Importación de Servicios Ajenos*) para la estrategia de generación de código (capítulo 7). El ejemplo se refiere a la importación del servicio Web para búsquedas de Google [82,83].

Para la definición de los modelos y metamodelos se ha utilizado el marco de trabajo *Eclipse Modeling Framework* (EMF [55]). EMF permite además la generación de un editor rudimentario de modelos. Este editor ha sido suficiente para la implementación de las ideas propuestas en la tesis.

A lo largo del ejemplo se muestran los extractos más importantes del código. El código completo se encuentra en el sitio Web http://toolthesis-iscrquinter.wikispaces.com.

A.1.1 Primer paso: importación del servicio ajeno de Google al Modelo de Servicios

El **primer paso** de este escenario consiste en la importación del servicio ajeno al PIM de Servicios. Esto se realiza mediante una importación *Texto-a-Modelo* de la interfaz del servicio. El listado A.1.1 muestra un extracto de la interfaz WSDL del servicio de búsqueda de Google, enmarcando y resaltando en colores azul, rojo y verde las operaciones del servicio ajeno a importar (`doGetCachedPage`, `doSpellingSuggestion`, `doGoogleSearch`) en el Modelo de Servicios.

Listado A.1.1 Operaciones a importar al servicio ajeno de Google

```
<?xml version="1.0" ?>
...
<portType name="GoogleSearchPort">
<operation name="doGetCachedPage">
<input message="typens:doGetCachedPage" />
<output message="typens:doGetCachedPageResponse" />
</operation>
<operation name="doSpellingSuggestion">
<input message="typens:doSpellingSuggestion" />
<output message="typens:doSpellingSuggestionResponse" />
</operation>
<operation name="doGoogleSearch">
<input message="typens:doGoogleSearch" />
<output message="typens:doGoogleSearchResponse" />
</operation>
</portType>
…
<service name="GoogleSearchService">
...
</service>
</definitions>
```

El resultado de la importación *Texto-A-Modelo* ha generado el PIM de Servicios que se muestra en el editor EMF de la Figura A.1.1, donde se enmarcan las operaciones importadas.

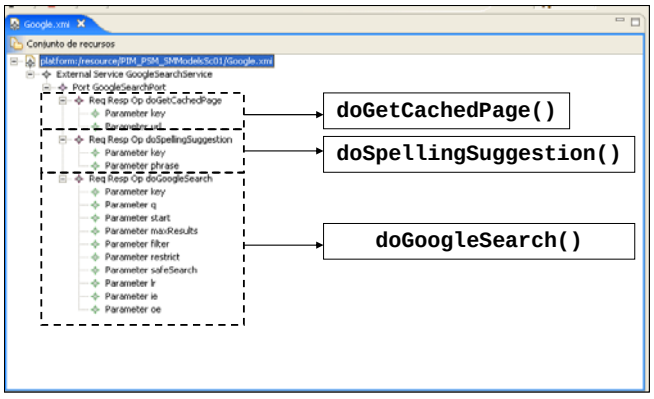


Figura A.1.1 PIM de Servicios para el servicio ajeno de Google

A.1.2 Segundo paso: transformación del PIM de servicios al PSM de servicios

En el **segundo paso**, el PIM de Servicios se corresponde a un PSM de Servicios para el marco de trabajo Axis y para la plataforma Java, mediante el conjunto de transformaciones *Modelo-A-Modelo* (implementadas en el lenguaje QVT operacional de Together [56]) que se muestran a continuación.

El listado A.1.2 muestra la regla de transformación `makeStub`. En esta regla por cada servicio ajeno (`ExternalService`) se crea un generador de stub (`ESStubGenerator`), inicializando su dirección WSDL. Invoca a la regla `makePort` para generar los puertos del servicio.

Listado A.1.2 Regla de Transformación `makeStub`

```
mapping makeStub(in model: kernel::ExternalService)
: psm_ws::ESStubGenerator
{
init {
        var allPorts := model.ports;
}
object {
        wsdlAddress := model.interfacelocation;
        iesfacport  += allPorts->collect(p|makePort(p));
}
}
```

La regla de transformación `makePort` está dada en el listado A.1.3. En esta regla, por cada puerto del servicio ajeno (`Port`) se crea una fachada (`IESFacadePort`). Invoca, a su vez, a la regla de transformación `makeOperation` para generar las operaciones del puerto.

Listado A.1.3 Regla de transformación `makePort`

```
mapping makePort(in port: kernel::Port)
:psm_ws::IESFacadePort {
init {
       var allOperations := port.operations;
}
object {
       namePort       := port.name;
       nameService    := port.service.name;
       esop += allOperations->collect(o|makeOperation(o));
}
}
```

Las reglas de transformación `makeOperation` y `makeParameter`, en conjunto, permiten la generación de cada una de las operaciones de la fachada, incluyendo sus parámetros. Se muestran en el listado A.1.4.

Listado A.1.4 Reglas de transformación `makeOperation` y `makeParameter`

```
mapping makeOperation(in operation: kernel::Operation): psm_ws::ESOperation {
init {
        var allParameters := operation.parameters;
}
object {
        name             := operation.name;
        esparameter += allParameters->collect(p|makeParameter(p));
        returnValueType  := operation.returnValueType;
}
}

mapping makeParameter(in parameter: kernel::Parameter): psm_ws::ESParameter {
object {
        name := parameter.name;
        type := parameter.type;
}
```

El PSM de Servicios generado, al aplicar estas transformaciones, se muestra en la Figura A.1.2 en el editor EMF. Se enmarca el generador de *stub* (`ESStubGenerator`) y cada operación generada.

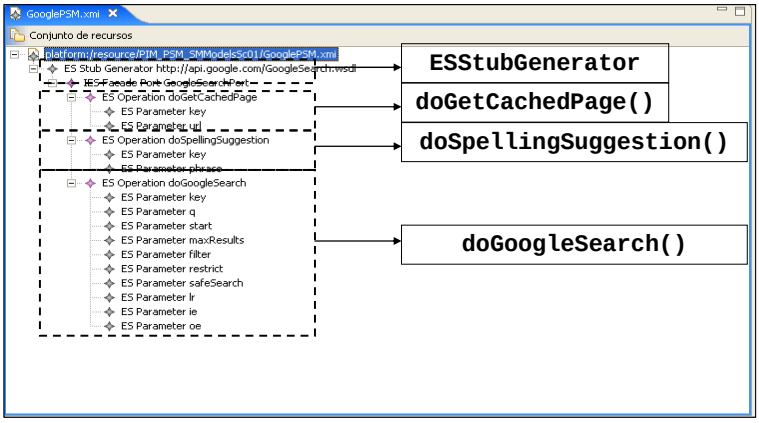


Figura A.1.2 PSM de servicios para las clases Axis y Java del servicio ajeno de Google

A.1.3 Tercer paso: generación de código final

En el **tercer paso**, la generación final de código, a partir del PSM de Servicios, está dada por un conjunto de transformaciones *Modelo-A-Texto* programadas en el lenguaje *MofScript*.

La transformación genera por cada puerto del servicio ajeno una fachada a partir de los valores almacenados en el generador de *stub*. La transformación que controla la generación de cada uno de los elementos de la fachada está dada en el listado A.1.5.

En esta transformación, un conjunto de transformaciones anidadas generan: la importación de las clases de soporte, el constructor de la clase, el código para el acceso a cada una de las operaciones del servicio ajeno y la operación de soporte para el acceso al puerto.

Listado A.1.5 Transformación Modelo-a-Texto principal para generar el *stub*

```
psmSM.IESFacadePort::generateJavaClass()
{
        file(self.nameService+self.namePort+"Facade.java");

        self.esstub.generateImports();
        self.initClass();
        self.generateInfAtt();
        self.generateConstructor();
        self.generateOperations();
        self.generateInfOps();
        endClass();
}
```

Listado A.1.6 Transformaciones anidadas para generar el *stub*

```
psmSM.ESStubGenerator::generateImports()
{
            'package ' self.nameSpace ';\n\n'

            'import java.rmi.RemoteException;\n'
            'import javax.xml.rpc.ServiceException;\n\n'
}
psmSM.IESFacadePort::initClass()
{
            'public class ' self.nameService self.namePort 'Facade\n'
            '{ \n'
}
psmSM.IESFacadePort::generateConstructor()
{
            'public ' self.nameService self.namePort 'Facade() throws ServiceException\n'
            '{ \n'
            '           initPort();\n'
            '} \n\n'
}
psmSM.IESFacadePort::generateInfAtt()
{
            'private ' self.nameService ' service = new ' self.nameService'Locator();\n'
            'private ' self.namePort ' port;\n\n'
}
psmSM.IESFacadePort::generateOperations()
{
            self.esop->forEach(op) {op.generateJavaOp() }
}
psmSM.IESFacadePort::generateInfOps()
{
            'private void initPort() throws ServiceException\n'
            '{\n'
            ' port = service.get' self.namePort '();\n'
            '}\n\n'
}
module::endClass()
{
            '}\n'
}
```

El detalle de cada una de las transformaciones anidadas está dado en el listado A.1.6. Después, a partir de los valores almacenados en el generador *stub*, se generan los mismos utilizando la herramienta `wsdl2java` del marco de trabajo de Axis. La llamada a la herramienta desde la transformación se muestra en el listado A.1.7.

Listado A.1.7 Llamando a `WSDL2Java` en la transformación Modelo-a-Texto

```
psmSM.ESStubGenerator::generateAxisStub()
{
        var wsdlAddress : String = self.wsdlAddress;

        java("adapters.wsdl2java.WSDL2JavaAdapter",
        "generateStubfrom",wsdlAddress,
        "C:/Documents and Settings/Ricardo Quintero/Mis documentos/
        Doctorado/Tesis/New Tesis/Tesis/wsTogether-EMF MDA/
        JavaAdapters2AXIS/bin/")
}
```

Finalmente, el código Java para la fachada está dado en el listado A.1.8. El código inicializa el acceso al servicio ajeno y cada operación delega su ejecución al código del *stub*-Axis. Se resaltan en colores las operaciones delegadas.

Listado A.1.8 Fachada Java hacia el servicio ajeno de Google

```
package GoogleSearch;
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;

public class GoogleSearchServiceGoogleSearchPortFacade
{
            private GoogleSearchService service = new GoogleSearchServiceLocator();
            private GoogleSearchPort port;

            public GoogleSearchServiceGoogleSearchPortFacade() throws ServiceException
            {
                        initPort();
            }

            public void doGetCachedPage(String key, String url) throws RemoteException
            {
                        port.doGetCachedPage(key,url);
            }

            public String doSpellingSuggestion(String key, String phrase)
            throws RemoteException
            {
                        return port.doSpellingSuggestion(key,phrase);
            }

            public GoogleSearchResult doGoogleSearch(String key, String q, int start,
            int maxResults, boolean filter, String restrict,
            boolean safeSearch, String lr, String ie, String oe) throws RemoteException
            {
                        return port.doGoogleSearch(key,q,start,maxResults,filter,restrict,
                        safeSearch,lr,ie,oe);
            }

            private void initPort() throws ServiceException
            {
                        port = service.getGoogleSearchPort();
            }
}
```

APÉNDICE 2

Generación de Servicios Propios Vistas de Clases de Negocio

En este apéndice se incluye un ejemplo de implementación para el escenario 2 (*Generación de Servicios Propios como vistas de Clases de Negocio*) de la estrategia de generación de código (capítulo 7).

Los modelos y metamodelos se han definido mediante el marco de trabajo del *Eclipse Modeling Framework* (EMF [55]). Las transformaciones de modelos utilizan los lenguajes QVT operacional de Together [56] y *MOFScript* [58]. El código final se genera con un compilador Java y con las herramientas `Java2WSDL` y `WSDL2Java` del marco de trabajo de Axis. Se incluyen extractos de código. El ejemplo completo se puede obtener en el sitio Web http://toolthesis-iscrquinter.wikispaces.com.

A.2.1 Modelo PIM de la Clase de Negocio

El ejemplo está dado para una Clase de Negocio `Producto` (del Diagrama de Clases) cuyas operaciones desean ser expuestas como operaciones de un servicio propio. La clase `Producto` se muestra en la Figura A.2.1. en el editor EMF. Se enmarcan sus operaciones principales (`getDescription`, `getPrice`, `setAuthor`, `setSold`, `getSold`).

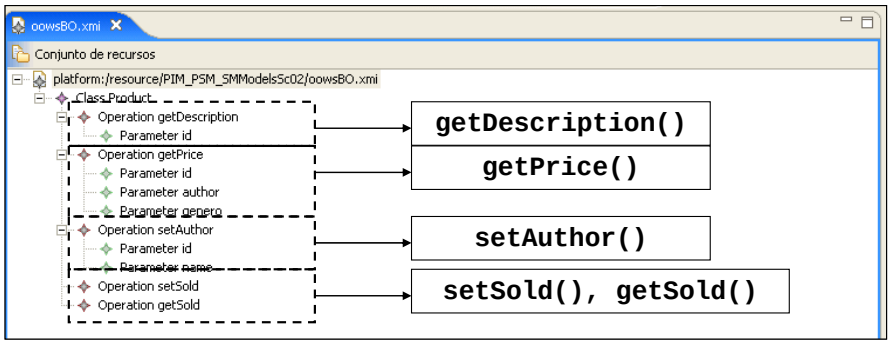


Figura A.2.1 Clase `Producto` como modelo EMF

A.2.2 Primero paso: generación del servicio propio a partir de la Clase de Negocio

En el **primer paso** de este escenario se parte de la Clase de Negocio anterior y, mediante el conjunto de transformaciones *Modelo-A-Modelo* que se explican a continuación, se genera el servicio propio que incluye las operaciones de la clase en el PIM de servicios.

La transformación `makeServiceAndPort` (Listado A.2.1), es la transformación central para obtener el servicio propio a partir de la Clase de Negocio. Asigna el nombre del servicio (`Service.name`) a partir del nombre de la clase (`Class.name`) y genera un puerto con todas las operaciones.

Listado A.2.1 Transformación para generar el servicio propio a partir de la Clase de Negocio

```
mapping makeServiceAndPort(in model: oowsBO::Class)
: kernel::OwnService {
init {
        var allOperationsOneWay  := model.operationClass
        ->select(o|o.returnType =  'void');
        var allOperationsReqResp := model.operationClass
        ->select(o|o.returnType <> 'void');
}

object {
        name := model.name + 'Service';
        ports +=
                object kernel::Port
                {
                        name := model.name + 'Port';
                        operations += allOperationsOneWay
                        ->collect(o|makeOperationOneWay(o));
                        operations += allOperationsReqResp
                        ->collect(o|makeOperationReqResp(o));
                }
        }
}
```

Las transformaciones para la generación de las operaciones del servicio propio (`makeOperationOneWay`, `makeOperationReqResp`, `makeParameter`) y sus parámetros, están dadas en el listado A.2.2. El tipo de operación a generar está dado por el tipo de valor de retorno. La clasificación se realiza en la transformación anterior.

Listado A.2.2 Transformación para generar el servicio propio a partir de la clase de Negocio

```
mapping makeOperationOneWay(in operation: oowsBO::Operation):
kernel::OneWayOp {
init {
        var allParameters := operation.ParameterClass;
}
object {
        name           := operation.name;
        returnValueType := operation.returnType;
        parameters     += allParameters->collect(p|makeParameter(p));
}
}
mapping makeOperationReqResp(in operation: oowsBO::Operation):
kernel::ReqRespOp {
init {
        var allParameters := operation.ParameterClass;
}
object {
        name           := operation.name;
        returnValueType := operation.returnType;
        parameters     += allParameters->collect(p|makeParameter(p));
}
}
mapping makeParameter(in parameter: oowsBO::Parameter):
kernel::Parameter {
object {
        name := parameter.name;
        type := parameter.type;
}
}
```

La Figura A.2.2 muestra el servicio propio `ProductService` generado a partir de la Clase de Negocio `Product` resultado de la transformación anterior. Se enmarcan las operaciones generadas (*OneWay* y *Request-Response*) que se han obtenido dependendiendo del tipo de retorno.

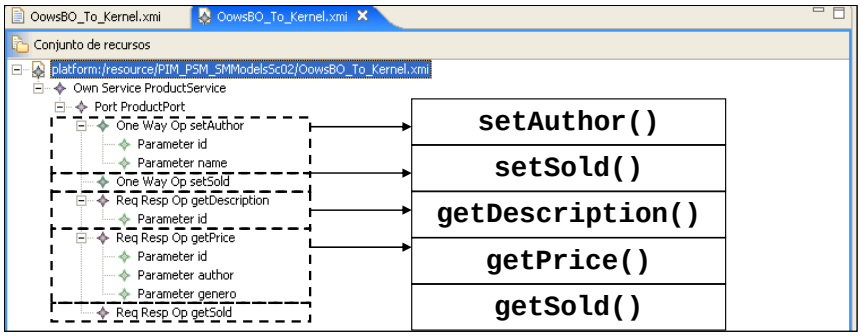


Figura A.2.2 Servicio propio `ProductService` generado a partir de `Product`

A.2.3 Segundo paso: generación de las clases PSM para interfaz WSDL, fachada y *skeletons*

En el **segundo paso**, se parte del PIM de servicio propio anterior para obtener el PSM de servicios para la generación de los *skeleton* Axis, interfaz WSDL y fachada Java.

La regla de transformación `makeSkeleton` genera el repositorio (`OSSkeletonAndWSDLGenerator`) con los valores necesarios para la generación del *skeleton* y la interfaz WSDL. Se muestra en el listado A.2.3.

Listado A.2.3 Regla de transformación `makeSkeleton`

```
mapping makeSkeleton(in model: kernel::OwnService):
psm_ws::OSSkeletonAndWSDLGenerator {
init {
        var firstPort     := model.getFirstPort();
        var allOperations := firstPort.operations;
        var serviceName   := model.name+firstPort.name;
}
object {
        wsdlFileName    := serviceName+'.wsdl';
        wsdlAddress     := 'http://localhost:8080/axis/services/'
        + serviceName;
        targetNameSpace := serviceName + 'Ns';
        packageName:= serviceName + 'Pck';
        classToDeploy   := serviceName+'.java';
        iosfacport      := object psm_ws::IOSFacadePort {
                                namePort := model.getFirstPort().name;
                                nameService:= model.name;
                                osop += allOperations
                                ->collect(o|makeOperation(o));
                         }
        }
}
```

Las reglas de transformación `makeOperation` y `makeParameter` generan las operaciones y los parámetros del servicios propio. Su implementación es semejante al listado A.1.4, por lo que no se repite aquí.

El PSM para la obtención del *skeleton* Axis, interfaz WSDL y la fachada Java generado por la transformación anterior está dado en la Figura A.2.3 Se enmarcan los valores generados en el repositorio y las operaciones del servicio propio.

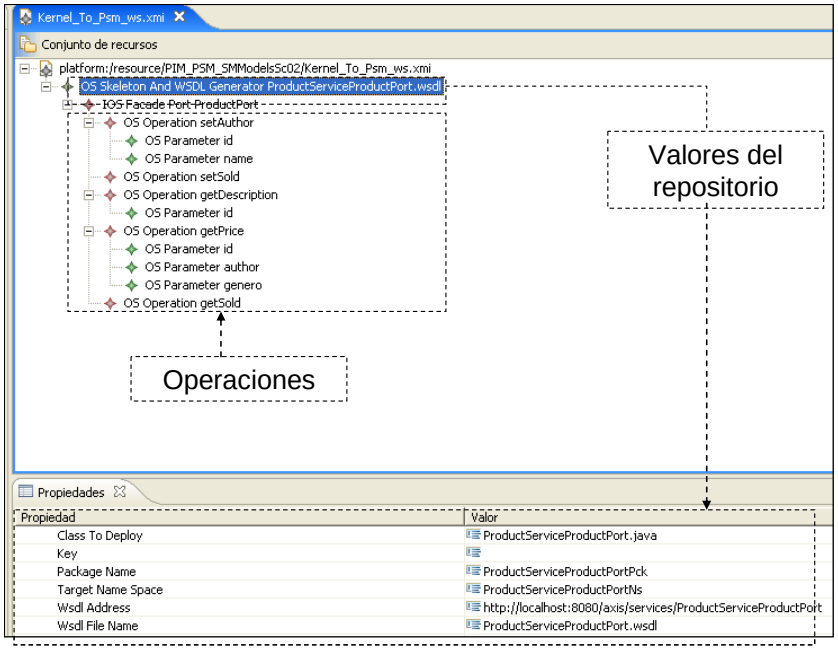


Figura A.2.3 PSM para generar skeleton y fachada

A.2.4 Tercer paso: generación de la clase fachada Java

En el **tercer paso**, un conjunto de transformaciones *Modelo-A-Texto* permiten la generación de la clase fachada Java, la cual posibilita a otros componentes software su acoplamiento con el servicio propio que se está generando. Esta fachada también se utilizará para la generación de código del servicio Web.

El listado A.2.4 muestra la transformación central para la generación de la fachada. Las partes principales de la misma obtienen: el inicio de la clase, atributos de infraestructura de la clase, el código de cada operación y el final de la clase. Cada uno de estos elementos se obtiene mediante transformaciones anidadas.

Listado A.2.4 Transformación central para generar clase fachada Java

```
psmSM.IOSFacadePort::generateJavaClass()
{
       file(self.nameService+self.namePort+"Facade.java");
       self.initClass();
       self.generateInfAtt();
       self.generateOperations();
       endClass();
}
```

Listado A.2.5 Transformación `generateJavaOp` para generar cada operación de la fachada

```
psmSM.OSOperation::generateJavaOp()
{
  self.generateSignature();
  self.generateBody();
}
psmSM.OSOperation::generateSignature()
{
  var nParams: Integer = self.osparameter.size();
  'public ' self.returnValueType ' ' self.name
  '('
  if (nParams > 0)
  {
    self.osparameter.first().generateParameter();
    self.osparameter->forEach(p)
    {
      if (position() >= 1)
      {
        ', ' p.generateParameter()
      }
    }
 }
 ')\n'
}
psmSM.OSOperation::generateBody()
{
  var nParams: Integer = self.osparameter.size();
  var rv: String = self.returnValueType;
  '{\n'
  if (!rv.equals("void"))
  {
     'return '
  }
  'bo.'self.name '('
  if (nParams > 0)
  {
    self.osparameter.first().generateBOParameter();
    self.osparameter->forEach(p)
    {
       if (position() >= 1)
       {
         ',' p.generateBOParameter()
       }
    }
  }
  ');\n'
  '}\n\n'
}
}
```

El listado A.2.5 muestra la transformación anidada `generateJavaOp`, invocada por `generateOperations` para obtener el código de cada operación de la fachada.

La clase fachada Java generada (`ProductServiceProductPortFacade`) a partir de esta transformación para el servicio propio del presente ejemplo, está dado por el listado A.2.6. La clase posee una referencia al objeto de negocio (resaltada en color rojo) y operaciones que delegan a las operaciones del mismo objeto (resaltadas en color azul).

Listado A.2.6 Clase fachada del servicio propio `ProductService`

```
public class ProductServiceProductPortFacade
{
private Product bo = new Product();

public void setAuthor(String id, String name)
{
        bo.setAuthor(id,name);
}
public void setSold()
{
        bo.setSold();
}
public String getDescription(String id)
{
        return bo.getDescription(id);
}
public int getPrice(String id, String author, String genero)
{
        return bo.getPrice(id,author,genero);
}
public Boolean getSold()
{
        return bo.getSold();
}
}
```

A.2.5 Cuarto paso: generación de la interfaz WSDL de la clase fachada Java

En el **cuarto paso** se hace uso de la herramienta `Java2WSDL` para obtener la interfaz WSDL de la clase fachada Java generada en el paso anterior. El listado A.2.7 muestra la interfaz generada resaltando sus operaciones.

Listado A.2.7 Interfaz WSDL para la fachada a `ProductService`

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://DefaultNamespace" ...>
        ...
   <wsdl:portType name="ProductServiceProductPortFacade">
      <wsdl:operation name="getDescription" parameterOrder="id">
         ...
      </wsdl:operation>
      <wsdl:operation name="setAuthor" parameterOrder="id name">
         ...
      </wsdl:operation>
      <wsdl:operation name="getPrice" parameterOrder="id author genero">
         ...
      </wsdl:operation>
      <wsdl:operation name="getSold">
         ...
      </wsdl:operation>
      <wsdl:operation name="setSold">
         ...
      </wsdl:operation>
         ...
   <wsdl:service name="ProductServiceProductPortFacadeService">
      ...
   </wsdl:service>
</wsdl:definitions>
```

A.2.6 Quinto paso: generación de las clases de enlace para el *skeleton*

En el **quinto paso**, con la interfaz WSDL antes generada y tomando como entrada en la herramienta `WSDL2Java` los valores del repositorio `OSSkeletonAndWSDLGenerator` del PSM de servicios, se generan las clases de enlace (*binding*) para la implementación del *skeleton*. El código completo de estas clases se encuentra en el sitio Web http://toolthesis-iscrquinter.wikispaces.com.

APÉNDICE 3

Agregación y Composición de Servicios

El tercer escenario (*Agregación y Composición de Servicios*) se ilustra en el siguiente apéndice. Su estrategia MDA genera código para el lenguaje BPEL, considerando como motor de ejecución al *Oracle BPEL Process Manager*. Se genera también una clase fachada que facilita el uso del servicio compuesto por los componentes software restantes.

Los modelos y metamodelos se han definido mediante el marco de trabajo *Eclipse Modeling Framework* (EMF [55]). Las transformaciones de modelos utilizan los lenguajes QVT operacional de Together [56] y *MOFScript* [58]. Una parte del código final se genera con un compilador Java, el resto con las herramientas `Java2WSDL` y `WSDL2Java` del marco de trabajo *Axis*.

A lo largo del ejemplo se muestran los extractos más importantes del código. El código completo se encuentra en el sitio Web http://toolthesis-iscrquinter.wikispaces.com.

A.3.1 Modelo PIM de agregación de servicios

El ejemplo está dado para el servicio propio `BestStoreService`, expuesto en el apartado 4.2.2.4 de esta tesis. Este servicio propio utiliza un par de servicios ajenos para su composición: `Amazon` y `BNQuoteService`. Su definición se muestra en el editor EMF de la Figura A.3.1 (dado el tamaño del modelo no se han expandido todas sus clases). Incluye el *Modelo de Servicios* y el *Modelo Dinámico*

de Composición de Servicios, ambos modelos están incluidos en uno sólo por la limitación del QVT operacional de Together, que en sus reglas de transformación sólo permite un modelo de entrada.

Se enmarcan las dos operaciones del servicio compuesto (`getBestStore` y `getBestPrice`) y se muestra el MDCS para la operación `getBestStore`. Se enmarcan también los servicios componente.

Figura A.3.1 Modelo de Servicio `BestStoreService` como modelo EMF

A.3.2 Primer paso: generación del PSM-BPEL con definición incompleta de variables

En el **primer paso**, se parte del Modelo de Servicios anterior y, mediante un conjunto de transformaciones *Modelo-A-Modelo*, se genera el modelo PSM-BPEL que incluye solamente las variables definidas en el *Modelo Dinámico de Composición de Servicios*, sin incluir las variables para los mensajes de entrada/salida para llamadas a operaciones de servicios componente.

La regla de transformación `makeSkeleton` se muestra en el listado A.3.1. Esta regla corresponde al servicio compuesto (`CompositeServ`) con el generador de *skeleton* e interfaz (`CSSkeletonAndWSDLGenerator`). El generador de *skeleton* es un repositorio de valores que incluye el nombre de la interfaz, su direc-

ción WSDL, el espacio de nombres destino, el nombre del paquete y la clase Java. Se muestra en color azul la generación del repositorio. Se inicializa también la clase fachada (`icsfacade`) al servicio compuesto. Se muestra en color rojo.

Listado A.3.1 Regla de transformación makeSkeleton

```
mapping makeSkeleton(in model: kernel::CompositeServ):
psm_bpel::CSSkeletonAndWSDLGenerator {
init {
        var firstPort     := model.getFirstPort();
        var allOperations := firstPort.operations;
        var serviceName   := model.name+firstPort.name;
}
object {
        wsdlFileName      := serviceName+'.wsdl';
        wsdlAddress       := 'http://localhost:8080/axis/services/'
                          + serviceName;
        targetNameSpace   := serviceName + 'Ns';
        packageName       := serviceName + 'Pck';
        classToDeploy     := serviceName+'.java';

        icsfacade         := object psm_bpel::ICSFacade {
                                 namePort   := model.getFirstPort().name;
                                 nameService:= model.name;
                                 icsoperation += allOperations
                                 ->collect(o|makeOperation(o));
                                 icsopFacade  += allOperations
                                 ->collect(o|makeFacOperation(o))
                          }
}
}
```

Las reglas `makeOperation` y `makeParameter` generan las operaciones para la fachada del servicio compuesto. Su definición está dada en el listado A.3.2.

Listado A.3.2 Reglas `makeOperation` y `makeParameter`

```
mapping makeOperation(in operation: kernel::Operation):
psm_bpel::ICSOperation
{
init {
        var allParameters := operation.parameters;
}
object {
        name         := operation.name;
        icsparameter += allParameters
                        ->collect(p|makeParameter(p));
        returnValueType := operation.returnValueType;
}
}

mapping makeParameter(in parameter: kernel::Parameter):
psm_bpel::ICSParameter
{
object {
        name := parameter.name;
        type := parameter.type;
        }
}
```

La regla de transformación `makeFacOperation` se muestra en el listado A.3.3. Esta regla genera el código BPEL a partir de la definición de la lógica de cada operación en el MDCS. Se resalta en color rojo la sección de generación de variables y en color azul se muestran extractos del código para generar la lógica de la operación (por razones de espacio no se muestra todo el detalle).

Listado A.3.3 Regla de transformación `makeFacOperation`

```
mapping makeFacOperation(in operation: kernel::Operation):
psm_bpel::ICSOpFacade
{
init {
        ...
}
object {
        ...
        bpeldefinition :=
        object psm_bpel::BPELSProcess {
                name               := operation.mdcsOpDef.name;
                BPELnsDef          := 'http://schemas.xmlsoap.org/ws/
                                       2003/03/business-process/';
                targetNSDef        := 'urn:'+operation.mdcsOpDef.mdcsOp.port.
                                       service.name + operation.mdcsOpDef.name;

                variable     += allVariables
                                ->collect(v|makeBPELVariable(v));

                variable     += object psm_bpel::Variable {
                                        name    := 'input';
                                        type    := 'messageType';
                                        typeDescription := operation.name+
                                        'RequestMessage';
                                  };

                variable     += object psm_bpel::Variable {
                                        name    := 'output';
                                        type    := 'messageType';
                                        typeDescription := operation.name+
                                        'ResponseMessage';
                                  };
                ...
                initSeq := object psm_bpel::Sequence {

                        activity += object psm_bpel::Receive {
                                ...
                        };

                        activity += allActions->collect(a|a.toActivity())
                                ->reject(a|a.oclIsTypeOf(psm_bpel::Activity));


                        activity += object psm_bpel::Reply {
                                ...
                        };
        }
}
```

El modelo PSM-BPEL con definición incompleta de variables que se ha generado por la transformación anterior se muestra en el editor EMF de la Figura A.3.2 (por el tamaño del modelo, algunas de sus clases no han sido expandidas).

Se enmarcan el generador de *skeleton*, y un ejemplo de PSM-BPEL generado para la operación `getBestStore`, incluyendo sus variables.

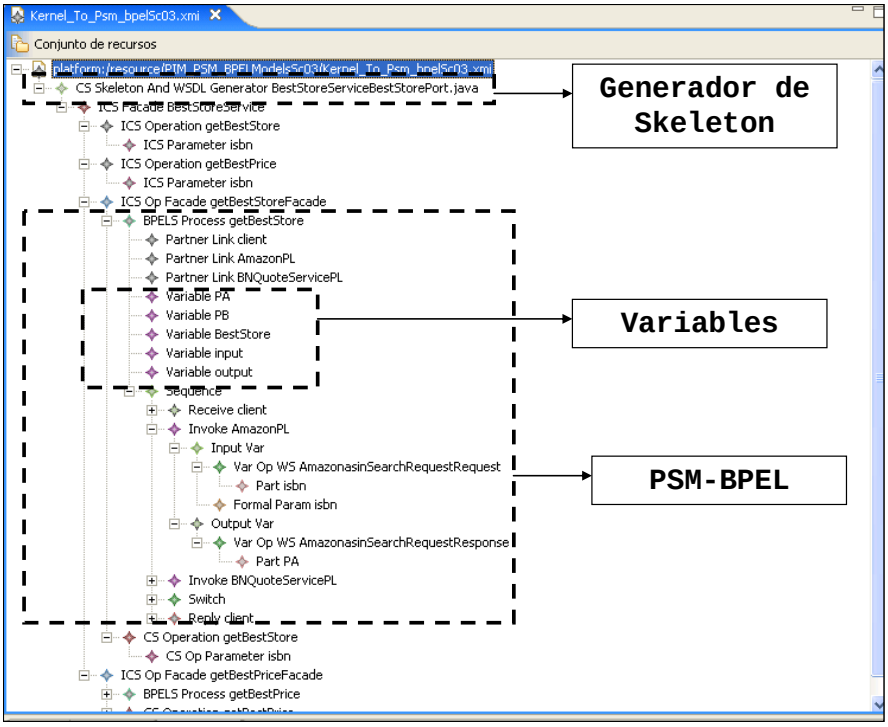


Figura A.3.2 El PSM-BPEL con definición incompleta de variables

A.3.3 Segundo paso: generación del PSM-BPEL con definición completa de variables

En el **segundo paso**, las variables faltantes de los mensajes de entrada/salida para las llamadas a las operaciones de los servicios componente, requieren de una transformación *Modelo-A-Modelo* adicional sobre el PSM-BPEL anterior.

Para cada una de las actividades del PSM-BPEL, que invoca a una operación de un servicio componente, un conjunto de transformaciones generan los mensajes de entrada y/o salida necesarios para la llamada. El listado A.3.4 muestra un ejemplo de este proceso para la operación `Invoke`, implementado mediante el *query* QVT operacional `generateVariables`.

Listado A.3.4 Query QVT operacional `Invoke::generateVariables`

```
query psm_bpel::Invoke::generateVariables()
:OrderedSet(psm_bpel::Variable)
{
if  self.outputVar.oclIsUndefined() then
OrderedSet {
        object psm_bpel::Variable {
                name            := self.inputVar.varOpWS.name;
                type            := 'messageType';
                typeDescription := '';
        }
}
else
OrderedSet {
        object psm_bpel::Variable {
                name            := self.inputVar.varOpWS.name;
                type            := 'messageType';
                typeDescription := '';
        },
        object psm_bpel::Variable {
                name := self.outputVar.varOpWS.name;
                type            := 'messageType';
                typeDescription := '';
        }
}
endif
}
```

Luego, se recorre el PSM-BPEL y se agregan las variables faltantes ya generadas con este tipo de query's. La transformación central para agregar estas variables es `addVariablesFaltantes`. Se muestra en el listado A.3.5.

Listado A.3.5 Regla de transformación `addVariablesFaltantes`

```
mapping addVariablesFaltantes(inout bpelmodel:psm_bpel::BPELProcess):
psm_bpel::BPELSProcess
{
init {
        var allActivities                := bpelmodel.initSeq.activity;

        var allActivitiesExceptFirst     := allActivities
                ->excluding(allActivities->first())
                ->asOrderedSet();
        var allActivitiesExceptFirstAndLast  := allActivitiesExceptFirst
                ->excluding(allActivitiesExceptFirst
                ->last())
                ->asOrderedSet();

        var allVariables                 :=
                        getVariables(allActivitiesExceptFirstAndLast);
        var allVariablesWithOutRepeated  :=
                        eliminateRepeated(allVariables)->asOrderedSet();

        bpelmodel.variable += allVariablesWithOutRepeated;
        result := bpelmodel.oclAsType(psm_bpel::BPELSProcess);
}
}
```

Aplicando las transformación anteriores se obtiene el PSM-BPEL con definición completa de variables que se muestra en la Figura A.3.3. Se enmarcan las variables faltantes generadas.

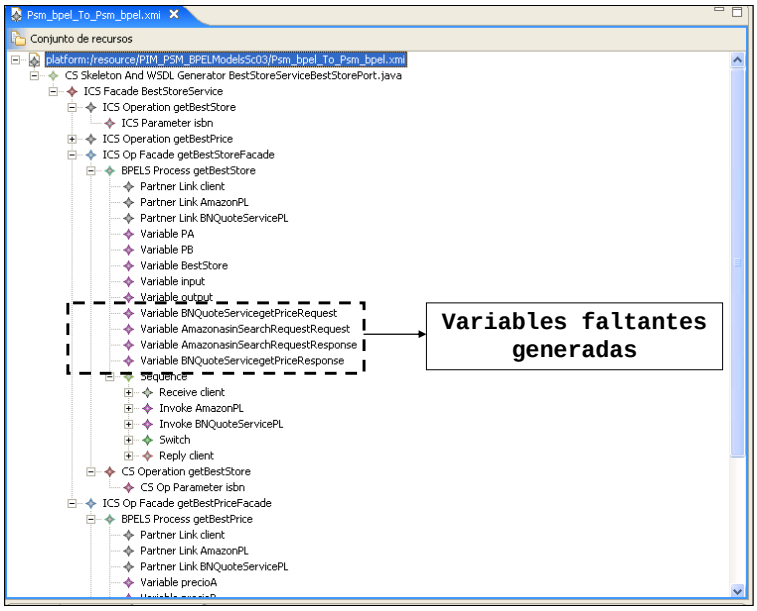


Figura A.3.3 PSM-BPEL con definición completa de variables

Partiendo de este modelo lo siguiente, en el **tercer paso** del escenario se generan los artefactos necesarios para la ejecución del servicio propio compuesto. A continuación se muestran cada una de las transformaciones *Modelo-A-Texto* que se requieren para el entorno *Oracle BPEL Process Manager*.

A.3.4 Tercer paso: generación de la operación componente como proceso BPEL

Las transformaciones *Modelo-A-Texto* centrales para la generación de cada una de las operaciones del servicio propio compuesto como proceso BPEL, están dadas en el listado A.3.6. La transformación `generateBPELProcess` controla las transformaciones anidadas restantes. La transformación `generateProcess-Begin` genera el tope del proceso, `generatexmlnsPartnerLinks` genera el código para el enlace con los servicios componente y `generateProcessEnd` genera la parte final del proceso.

Listado A.3.6 Transformaciones para generar las operaciones como proceso BPEL

```
psm_bpel.BPELSProcess ::generateBPELProcess()
  {
        self.generateProcessBegin()
        self.generatePartnerLinksSection()
        self.generateVariablesSection()
        self.generateBodySection()
        self.generateProcessEnd()
  }
psm_bpel.BPELSProcess::generateProcessBegin()
  {
    '<process name="' self.name '"' newline(1)
    tab(2) 'targetNamespace="' self.targetNSDef '"' newline(1)
    tab(2) 'xmlns:tns="' self.targetNSDef '"' newline(1)
    tab(2) 'xmlns="' self.BPELnsDef '"' newline(1)
    tab(2) 'xmlns:xsd="http://www.w3.org/2001/XMLSchema"' newline(1)
    tab(2) 'xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"'
           newline(1)
    self.generatexlmnsPartnerLinks() '>' newline(1)
  }
psm_bpel.BPELSProcess::generatexlmnsPartnerLinks()
  {
    self.partnerlink->forEach(p:psm_bpel.PartnerLink|not p.name.endsWith('client'))
    {
      tab(2) 'xmlns:' p.partnerRole '="' p.addressPort '"' newline(1)
    }
  }

psm_bpel.BPELSProcess::generateProcessEnd()
  {
         '</process>' newline(1)
  }
```

Listado A.3.7 Transformación para generar el cuerpo del Proceso BPEL

```
psm_bpel.BPELSProcess::generateBodySection()
{
         generateBodySectionBegin()
         self.initSeq.activity->forEach(a) { a.generateActivity() }
         generateBodySectionEnd()
}

psm_bpel.Activity::generateActivity()
{
 if (self.oclIsKindOf(psm_bpel.Invoke))
          self.generateInvoke()
 else
 if (self.oclIsKindOf(psm_bpel.Reply))
          self.generateReply()
 else
 if (self.oclIsKindOf(psm_bpel.Receive))
          self.generateReceive()
 else
 if (self.oclIsKindOf(psm_bpel.Assign))
          self.generateAssign()
 else
 if (self.oclIsKindOf(psm_bpel.Switch))
          self.generateSwitch()
 else
 if (self.oclIsKindOf(psm_bpel.Sequence))
          self.generateSequence()
}
}
```

El listado A.3.7 muestra la transformación para generar el cuerpo del proceso BPEL. Para cada una de las actividades del proceso se define una transformación

que genera su código. El listado A.3.8 muestra, como ejemplo, la transformación `generateReceive` para para la operación `Receive`.

Listado A.3.8 Transformación `generateReceive`

```
psm_bpel.Receive::generateReceive()
{
        if (self.partnerlink.endsWith("client"))
        {
                '<receive partnerLink="'  self.partnerlink '"' newline(1)
                tab(1) 'portType="tns:'   self.portType '"' newline(1)
                tab(1) 'operation="'      self.operation '"' newline(1)
                tab(1) 'variable="'       self.variable '"' newline(1)
                tab(1) 'createInstance="' self.createInstance '" />' newline(1)
        }
        else
        {
                '<receive partnerLink="' self.partnerlink '"' newline(1)
                tab(1) 'portType="' self.portType '"' newline(1)
                tab(1) 'operation="' self.operation '"' newline(1)
                tab(1) 'variable="' self.variable newline(1)
                tab(1) 'createInstance="' self.createInstance '" />' newline(1)
        }
}
```

Extractos del código BPEL generado con esta transformación para el ejemplo, están dados en los listados A.3.9 al A.3.12. Muestra el tope del proceso, los *partnerlinks*, las variables y un extracto de la secuencia de actividades.

Listado A.3.9 El tope del Proceso BPEL

```
<process name="getBestStore"
targetNamespace="urn:BestStoreServicegetBestStore"
xmlns:tns="urn:BestStoreServicegetBestStore"
xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:AmazonService="http://levitico:8080/AmazonWebModule/services/AmazonWSPort"
xmlns:BNQuoteServiceService="http://levitico:8080/BNWebModule/services/BNWSPort"
>
```

Listado A.3.10 La sección PartnerLinks

```
<partnerLinks>
        <partnerLink name="client"
                partnerLinkType="tns:getBestStore_clientLT"
                myRole="getBestStoreService"
        />
        <partnerLink name="AmazonPL"
                partnerLinkType="AmazonService:AmazonLT"
                partnerRole="AmazonService"
        />
        <partnerLink name="BNQuoteServicePL"
                partnerLinkType="BNQuoteServiceService:BNQuoteServiceLT"
                partnerRole="BNQuoteServiceService"
        />
</partnerLinks>
```

Listado A.3.11 La sección de variables

```
<variables>
    <variable name="PA“ type="xsd:float"/>
    <variable name="PB“ type="xsd:float"/>
    <variable name="BestStore“ type="xsd:string"/>
    <variable name="input“ messageType="tns:getBestStoreRequestMessage"/>
    <variable name="output“ messageType="tns:getBestStoreResponseMessage"/>
    <variable name="BNQuoteServicegetPriceRequest“
    messageType="BNQuoteServicegetPriceRequestMessage"/>
    <variable name="AmazonasinSearchRequestRequest“
    messageType="AmazonasinSearchRequestRequestMessage"/>
    <variable name="AmazonasinSearchRequestResponse"
    messageType="AmazonasinSearchRequestResponseMessage"/>
    <variable name="BNQuoteServicegetPriceResponse"
    messageType=""/>
</variables>
```

Listado A.3.12 La secuencia de actividades

```
<sequence>
         <receive partnerLink="client"
         portType="tns:getBestStorePT"
         operation="getBestStore"
         variable="input"
         createInstance="yes" />

         <assign>
         <copy>
                  <from variable="input" part="payload"
                  query="/tns:getBestStoreRequest/tns:isbn"></from>
                  <to variable="AmazonasinSearchRequestRequest"
                  part="isbn"/>
         </copy>
         </assign>

         <invoke partnerLink="AmazonPL"
                  portType="AmazonService:AmazonJavaWS"
                  operation="asinSearchRequest"
                  inputVariable="AmazonasinSearchRequestRequest"
                  outputVariable="AmazonasinSearchRequestResponse" />
         ...
</sequence>
```

A.3.5 Tercer paso: generación de la interfaz WSDL de la operación componente

La interfaz WSDL de cada operación se obtiene mediante la transformación *Modelo-A-Texto* del listado A.3.13. La transformación incluye transformaciones anidadas para la generación del tope de la interfaz (`generateDefinitionsTope`), los tipos de datos (`generateTypes`), los mensajes (`generateMessages`), las interfaces WSDL importadas (`generateImports`), los puertos (`generatePort`), los *partnerlinks* (`generatePartnerLinks`) y el final (`gene-`

`rateDefinitionsBottoms`). Por razones de espacio sólo se muestra el código para generar el puerto y los *partnerLinks.*

Listado A.3.13 Transformación para generar la interfaz WSDL de cada operación componente

```
psm_bpel.BPELSProcess ::generateWSDL()
{
file(self.name+".wsdl");
self.generateWSDLBegin()
self.generateDefinitons()
}
...
psm_bpel.BPELSProcess::generateDefinitons()
{
self.generateDefinitionsTope()
self.generateTypes()
self.generateMessages()
self.generateImports()
self.generatePort()
self.generatePartnerLinks()
self.generateDefinitionsBottom()
}
...
psm_bpel.BPELSProcess::generatePort()
{
'<portType name="' self.name+'PT' '">' newline(1)
tab(1) '<operation name="' self.name '">' newline(1)
tab(2) '<input message="tns:' self.name 'RequestMessage"/>' newline(1)
tab(2) '<output message="tns:' self.name 'ResponseMessage"/>' newline(1)
tab(1) '</operation>' newline(1)
'</portType>' newline(1)
}
...
psm_bpel.BPELSProcess::generatePartnerLinks()
{
  self.partnerlink->forEach(p:psm_bpel.PartnerLink|p.name.equals("client"))
  {
    '<plnk:partnerLinkType name="' p.partnerLinkType '">' newline(1)

    if (p.myRole.trim().size() > 0)
    {
      tab(1) '<plnk:role name="' p.myRole.trim() '">' newline(1)
      tab(2) '<plnk:portType name="tns:' self.name+'PT' '"/>' newline(1)
      tab(1) '</plnk:role>' newline(1)
    }

    if (p.partnerRole.trim().size() > 0)
    {
      tab(1) '<plnk:role name="' p.partnerRole.trim() '">' newline(1)
      tab(2) '<plnk:portType name="' p.namePort '"/>' newline(1)
      tab(1) '</plnk:role>' newline(1)
    }

    '</plnk:partnerLinkType>' newline(1)
  }
}
```

Como ejemplo de interfaz WSDL generada, el listado A.3.14 muestra la correspondiente a la operación `getBestStore`. Se resaltan los elementos generados por la transformación anterior.

Listado A.3.14 Interfaz WSDL generada para la operación `getBestStore`

```
<?xml version="1.0" encoding="utf-8" ?>
<definitions name="getBestStore"
                    targetNamespace="urn:BestStoreServicegetBestStore"
                    xmlns:tns="urn:BestStoreServicegetBestStore"
                    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                    xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
                    xmlns="http://schemas.xmlsoap.org/wsdl/">

          <types>
          <schema targetNamespace="urn:BestStoreServicegetBestStore"
          attributeFormDefault="unqualified"
          elementFormDefault="qualified" xmlns="http://www.w3.org/2001/XMLSchema">

          <element name="getBestStoreRequest">
                    <complexType>
                    <sequence>
                              <element name="isbn" type="string"/>
                    </sequence>
                    </complexType></element>

          <element name="getBestStoreResponse">
                    <complexType>
                    <sequence>
                              <element name="result" type="string"/>
                    </sequence>
                    </complexType></element>
          </schema>
          </types>

          <message name="getBestStoreRequestMessage">
                    <part name="payload" element="tns:getBestStoreRequest" />
          </message>

          <message name="getBestStoreResponseMessage">
                    <part name="payload" element="tns:getBestStoreResponse" />
          </message>

          <portType name="getBestStorePT">
                    <operation name="getBestStore">
                              <input message="tns:getBestStoreRequestMessage"/>
                              <output message="tns:getBestStoreResponseMessage"/>
                    </operation>
          </portType>

          <plnk:partnerLinkType name="getBestStore_clientLT">
                    <plnk:role name="getBestStoreService">
                    <plnk:portType name="tns:getBestStorePT"/>
          </plnk:role>
</plnk:partnerLinkType>
</definitions>
```

A.3.6 Tercer paso: generación de la interfaz envolvente WSDL de los servicios componente

Para cada servicio componente es necesario definir una interfaz envolvente (*wrapper*) WSDL. Esta interfaz se genera con la transformación *Modelo-A-Texto* del Listado A.3.15. La transformación identifica los *partnerLink* del proceso BPEL y genera para cada uno de ellos las secciones de la interfaz envolvente: el inicio (`generateWSDLWrapperBegin`), la importación (`generateWSDLWrapperIm-`

port), los partnerLinks (generateWSDLWrapperPartnerLinks) y el final (generateWSDLWrapperEnd).

Listado A.3.15 Transformación para generar la interfaz envolvente WSDL a cada servicio componente

```
psm_bpel.BPELSProcess::generateAllWSDLWrappers()
{
self.partnerlink->forEach(p:psm_bpel.PartnerLink){
if (p.name <> 'client')
{
         file(p.bpelprocess.name+p.name+".wsdl");
         p.generateWSDLWrapperBegin()
         p.generateWSDLWrapperImport()
         p.generateWSDLWrapperPartnerLinks()
         p.generateWSDLWrapperEnd()
}
}
}
psm_bpel.PartnerLink::generateWSDLWrapperBegin()
{
'<?xml version="1.0" encoding="utf-8"?>' newline(1)
'<definitions  name="'self.partnerLinkType '"' newline(1)
tab(1) 'targetNamespace="' self.addressPort '"' newline(1)
tab(1) 'xmlns:tns="' self.addressPort '"' newline(1)
tab(1) 'xmlns="http://schemas.xmlsoap.org/wsdl/"' newline(1)
tab(1) 'xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"'
        newline(1)
'>' newline(1)
}
psm_bpel.PartnerLink::generateWSDLWrapperImport()
{
tab(1) '<import location="' self.wsdlAddress '"/>' newline(1)
}
psm_bpel.PartnerLink::generateWSDLWrapperPartnerLinks()
{
tab(1) '<plnk:partnerLinkType name="' self.partnerLinkType '">' newline(1)
tab(2) '<plnk:role name="' self.partnerRole '">' newline(1)
tab(3) '<plnk:portType name="tns:' self.namePort '" />' newline(1)
// Limitado a una operacion por puerto ----
tab(2) '</plnk:role>' newline(1)
tab(1) '</plnk:partnerLinkType>' newline(1)
}

psm_bpel.PartnerLink::generateWSDLWrapperEnd(){
'</definitions>' newline(1)
}
```

El Listado A.3.16 muestra un ejemplo de interfaz envolvente para el servicio Amazon.com.

Listado A.3.16 Interfaz envolvente WSDL generada para el servicio Amazon

```
<?xml version="1.0" encoding="utf-8"?>
<definitions  name="AmazonLT"
targetNamespace="http://levitico:8080/AmazonWebModule/services/AmazonWSPort"
xmlns:tns="http://levitico:8080/AmazonWebModule/services/AmazonWSPort"
        xmlns="http://schemas.xmlsoap.org/wsdl/"
        xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">

        <import location=
        "http://levitico:8080/AmazonWebModule/services/AmazonWSPort?wsdl"/>

        <plnk:partnerLinkType name="AmazonLT">
                <plnk:role name="AmazonService">
                        <plnk:portType name="tns:AmazonWSPort" />
                </plnk:role>
        </plnk:partnerLinkType>
</definitions>
```

A.3.7 Tercer paso: generación del descriptor de procesos

Cada motor de ejecución BPEL requiere de un descriptor de procesos para los servicios compuestos que hospedará, por lo que es necesario definir una transformación *Modelo-A-Texto* para su generación. El Listado A.3.17 muestra la transformación.

Listado A.3.17 Transformación para la generación del Descriptor de Proceso

```
psm_bpel.BPELSProcess::generatebpelxml()
{
file(self.name+"bpel.xml");
self.generatebpelxmlBegin()
self.generateWSDLlocations()
self.generatebpelxmlEnd()
}
psm_bpel.BPELSProcess::generatebpelxmlBegin()
{
'<?xml version="1.0" encoding="UTF-8" ?>' newline(1)
'<BPELSuitcase>' newline(1)
'<BPELProcess src="' self.name+'.bpel"' ' id="' self.name 'Process">' newline(1)
}
psm_bpel.BPELSProcess::generateWSDLlocations()
{
'<partnerLinkBindings>' newline(1)
self.partnerlink->forEach(p:psm_bpel.PartnerLink) {
        tab(1) '<partnerLinkBinding name="' p.name '">' newline(1)
        tab(2) '<property name ="wsdlLocation">' newline(1)
        tab(3)

        if (p.name = 'client')
                self.name+'.wsdl'
        else
                'services/' p.name '.wsdl'
                newline(1)
                tab(2) '</property>' newline(1)
                tab(1) '</partnerLinkBinding>' newline(1)
}
'</partnerLinkBindings>' newline(1)
}
psm_bpel.BPELSProcess::generatebpelxmlEnd(){
'</BPELProcess>' newline(1)
 '</BPELSuitcase>' newline(1)
}
```

Las transformaciones anidadas de la transformación generan: el tope del descriptor (`generatebpelXMLBegin`), las direcciones de las interfaces WSDL de los *partnerlinks* (`generateWSDLlocations`) y el final del descriptor (`generatebpelXMLEnd`).

El Listado A.3.18 ofrece el descriptor de procesos generado para el proceso `getBestoreProcess`. Se resaltan las direcciones de las interfaces WSDL de los *partnerlinks*.

Listado A.3.18 Descriptor de proceso generado para `getBestoreProcess`

```
<?xml version="1.0" encoding="UTF-8" ?>
<BPELSuitcase>
<BPELProcess src="getBestStore.bpel" id="getBestStoreProcess">
<partnerLinkBindings>
        <partnerLinkBinding name="client">
                <property name ="wsdlLocation">
                        getBestStore.wsdl
                </property>
        </partnerLinkBinding>
        <partnerLinkBinding name="AmazonPL">
                <property name ="wsdlLocation">
                        services/AmazonPL.wsdl
                </property>
        </partnerLinkBinding>
        <partnerLinkBinding name="BNQuoteServicePL">
                <property name ="wsdlLocation">
                        services/BNQuoteServicePL.wsdl
                </property>
        </partnerLinkBinding>
</partnerLinkBindings>
</BPELProcess>
</BPELSuitcase>
```

A.3.8 Tercer paso: generación del descriptor y el constructor del proyecto

Para la construcción e instalación del código generado se utilizó el Ambiente Integrado de Desarrollo (AID) *Oracle BPEL PM Designer* y el compilador *obant*. Para que los servicios compuestos puedan ser utilizados en este AID y sea posible su construcción e instalación en el servidor de aplicaciones, es necesario la generación de un descriptor de proyecto y un constructor. Este par de artefactos se generan con las transformaciones *Modelo-A-Texto* de los Listados A.3.19 y A.3.20.

La transformación del listado A.3.19 toma el nombre de la definición del proyecto (`BPELSProcess.name`) y genera el fichero XML (`.project`) del descriptor del proyecto.

Listado A.3.19 Transformación para la generación del descriptor del proyecto

```
psm_bpel.BPELSProcess::generateproject()
{
file("." + self.name + "project");

'<?xml version="1.0" encoding="UTF-8"?>' newline(1)
'<projectDescription>' newline(1)
tab(1) '<name>'self.name '</name>' newline(1)
tab(1) '<comment></comment>' newline(1)
tab(1) '<projects>' newline(1)
tab(1) '</projects>' newline(1)
tab(1) '<buildSpec>' newline(1)
tab(2) '<buildCommand>' newline(1)
tab(3) '<name>com.oracle.bpel.designer.bpelz.BPELBuilder</name>' newline(1)
tab(3) '<arguments>' newline(1)
tab(3) '</arguments>' newline(1)
tab(2) '</buildCommand>' newline(1)
tab(1) '</buildSpec>' newline(1)
tab(1) '<natures>' newline(1)
tab(2) '<nature>com.oracle.bpel.designer.bpelz.BPELNature</nature>' newline(1)
tab(1) '</natures>' newline(1)
'</projectDescription>' newline(1)
}
}
```

Igualmente, la transformación del listado A.3.20 toma el nombre del proceso y genera el constructor (`build.xml`).

Listado A.3.20 Transformación para la generación del constructor del proyecto

```
psm_bpel.BPELSProcess::generatebuildxml()
{
file(self.name + "build.xml");
'<?xml version="1.0"?>' newline(1)
'<project name="' self.name '" default="main" basedir=".">' newline(1)
    tab(1) '<property name="deploy" value="default"/>' newline(1)
    tab(1) '<property name="rev" value="1.0"/>' newline(1)
    tab(1) '<target name="main">' newline(1)
    tab(2) '<bpelc  home="${home}"' newline(1)
    tab(4) 'rev="${rev}"' newline(1)
    tab(4) 'deploy="${deploy}"' newline(1)
    tab(4) '/>' newline(1)
    tab(2) '</target>' newline(1)
'</project>' newline(1)
}
}
```

Finalmente los listados A.3.21 y A.3.22 muestra el descriptor y el constructor generados para el proceso `getBestoreProcess`.

Listado A.3.21 Descriptor del proyecto generado

```
<?xml version="1.0" encoding="UTF-8"?>
<projectDescription>
        <name>getBestStore</name>
        <comment></comment>
        <projects>
        </projects>
        <buildSpec>
                <buildCommand>
                        <name>com.oracle.bpel.designer.bpelz.BPELBuilder</name>
                        <arguments>
                        </arguments>
                </buildCommand>
        </buildSpec>
        <natures>
                <nature>com.oracle.bpel.designer.bpelz.BPELNature</nature>
        </natures>
</projectDescription>
```

Listado A.3.22 Constructor del proyecto generado

```
<?xml version="1.0"?>
<project name="getBestStore" default="main" basedir=".">
<property name="deploy" value="default"/>
<property name="rev" value="1.0"/>
<target name="main">
<bpelc  home="${home}"
rev="${rev}"
deploy="${deploy}"
/>
</target>
</project>
```

A.3.9 Últimos pasos: generación de fachadas para el servicio compuesto y operaciones componente

La generación de las fachadas Java para el servicio compuesto y las operaciones componente es realizada mediante transformaciones *Modelo-A-Texto* desde el PSM con definición completa de variables. Por ser semejante a las transformaciones del escenario 2 se han omitido. Véase el apéndice 2 para un ejemplo de como se implementan estas transformaciones y el resultado que obtienen.

APÉNDICE 4

Especialización de Servicios

Este apéndice muestra un ejemplo de implementación del cuarto escenario, *Especialización de servicios*. Igual que en los apéndices anteriores contiene un ejemplo de aplicación de la estrategia MDA para el escenario mencionado. Como ha sido presentado en el capítulo 7, el escenario se implementa en dos etapas, siendo la primera (*Transformación de la especialización de servicios en agregación de servicios*) la que se presenta aquí, puesto que la segunda es igual al escenario tres, del cual ya ha sido presentado un ejemplo en el Apéndice 3.

Los modelos y metamodelos se han definido mediante el marco de trabajo del *Eclipse Modeling Framework* (EMF [55]). Las transformaciones de modelos utilizan los lenguajes QVT operacional de Together [56] y *MOFScript* [58].

A lo largo del ejemplo se muestran los extractos más importantes del código. El código completo se encuentra en el sitio Web http://toolthesis-iscrquinter.wikispaces.com.

A.4.1 Modelo PIM de especialización de servicios

El ejemplo está dado para la especialización del servicio ajeno `TravelService` en el servicio propio `TravelHotelService` del apartado 4.2.2.2. En el servicio `TravelService` se especializa la operación `getTripDetails`. La Figura A.4.1 muestra el modelo EMF. El servicio base y el servicio derivado se resaltan.

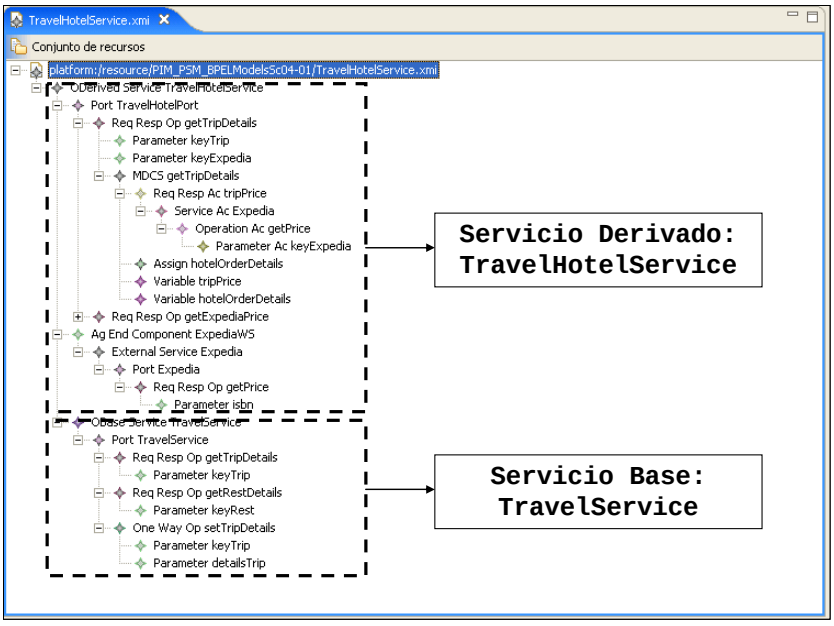


Figura A.4.1 Modelo EMF de especialización de `TravelService` en `TravelHotelService`

A.4.2 Primer y segundo paso: transformación de especialización a composición

La transformación del servicio especializado a servicio compuesto se realiza mediante una transformación *Modelo-A-Modelo*. Los **dos primeros pasos** mencionados en el apartado 7.2.4 para esta transformación (desde la *Transformación de la especialización en agregación* hasta la *Especialización de operaciones*) se incluyen en la programación de este apartado.

En el **primer paso** se copia el servicio derivado como un servicio compuesto, al que se agrega como servicio componente el servicio base. Este proceso es realizado por la regla de transformación `makeComposite`, mostrada en el listado A.4.1. Esta regla recibe en su entrada el servicio derivado (`ODerivedService`) y agrega como componente el servicio base (`model.obaservice`). Se resalta en color azul la sección de la regla que realiza el copiado.

Listado A.4.1 Regla `makeComposite`

```
mapping makeComposite(in model: kernel::ODerivedService)
: kernel::CompositeServ
{
object {
         name             := model.name;
         ports            += model.ports
                              ->collect(p|makePort(p))
                              ->asOrderedSet();

         agEndComponents  := model.agEndComponents;
         agEndComponents += object kernel::AgEndComponent {
                                   name      := '_BS_';
                                   component :=
                                  convertToOwnService(model.obaseservice)
         };
         }
}
```

La Figura A.4.2 muestra el modelo resultante al convertir el servicio derivado en una composición. Se enmarca el servicio derivado (`TravelHotelService`) y el servicio base (`TravelService`) como componente.

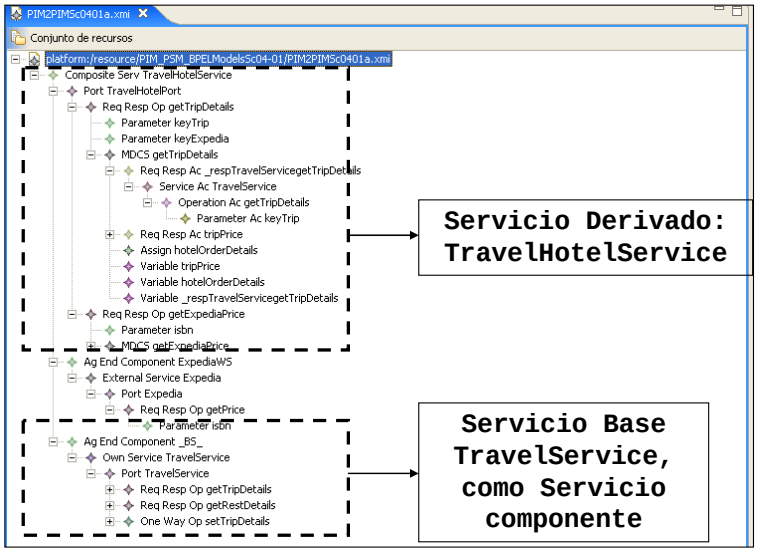


Figura A.4.2 Servicio compuesto obtenido a partir del servicio especializado

A la transformación anterior le resta una transformación adicional para especializar las operaciones (**paso 2** del escenario). Esto se lleva a cabo con las transformaciones *toNewMDCS<Action>Op* y *toOneActionMDCSfor<Actio>Op*. Del primer tipo el listado A.4.2 muestra el ejemplo para la operación `ReqRespOp`. Este

representa el caso en el que se refina la signatura de la operación, así como su lógica. La regla transforma el MDCS del servicio base en una nuevo que llama a la acción original, agregando las nuevas acciones de la operación especializada. Se resalta la sección de la regla que lleva a cabo esta tarea.

Listado A.4.2 Regla de transformación `toNewMDCSforReqrRespOp`

```
mapping kernel::MDCS::toNewMDCSforReqRespOp():kernel::MDCS
{
init {
        var nameOperationSpec := self.mdcsOp.name;
        var opBaseService     := self.mdcsOp.port.service.
                                 oclAsType(kernel::ODerivedService).
                                 obaseservice.ports->first().
                                 operations->select(name=nameOperationSpec)
                                 ->first();
        var typeOpBaseService := opBaseService.returnValueType;
        var serviceAc         := object kernel::ServiceAc {
                                         name         := opBaseService.port.service.name;
                                         operationsAc := object kernel::OperationAc {
                                                                 name        := opBaseService.name;
                                                                 parameterAc := opBaseService.parameters
                                                                 ->collect(p|
                                                                 object kernel::ParameterAc {
                                                                         name := p.name
                                                                 })->asOrderedSet()
                                         }
                                        };
        var newAction         :=  if opBaseService.oclIsKindOf(kernel::ReqRespOp) then
                                         opBaseService.oclAsType(kernel::ReqRespOp).toReqRespAc(serviceAc)
                                 else
                                         object kernel::ReqRespAc {
                                                 varName := 'Accion ReqResp no conocida'
                                         }
                                 endif
        }
        object kernel::MDCS {
                 name        := self.name;
                 returnValue := self.returnValue;
                 variable    := self.variable;
                 variable    += object kernel::Variable {
                                  name := '_resp' + serviceAc.name + serviceAc.operationsAc.name;
                                  type := typeOpBaseService
                                  };
                 action      += newAction;
                 action      += self.action;
        }
}
```

Del segundo tipo el listado A.4.3 muestra el ejemplo para la operación `ReqRespOp` del servicio base que no se especializa. Esta operación genera un MDCS en el servicio derivado que posee una acción que invoca a la operación original del servicio base. La regla de transformación `toOneActionMDCSforReqRespOp` crea la operación en el servicio derivado y agrega un MDCS con una sola acción que invoca a la operación original en el servicio base.

Listado A.4.3 Regla de transformación `toOneActionMDCSforReqRespOp`

```
mapping kernel::ReqRespOp::toOneActionMDCSforReqRespOp():kernel::MDCS
{
init {
var serviceAc    := object kernel::ServiceAc {
                         name  := self.port.service.name;
                         operationsAc := object kernel::OperationAc {
                         name  := self.name;
                         parameterAc := self.parameters
                                  ->collect(p|
                                  object kernel::ParameterAc {
                                           name := p.name
                                  })->asOrderedSet()
                         }
                     };
    var newAction         :=
         if self.oclIsKindOf(kernel::ReqRespOp) then
                  self.oclAsType(kernel::ReqRespOp).toReqRespAc(serviceAc)
         else
                  object kernel::ReqRespAc {
                           varName := 'Accion unica ReqResp no conocida'
                  }
         endif
}

object kernel::MDCS {
         name        := self.name;
         returnValue := '_resp' + serviceAc.name
         + serviceAc.operationsAc.name;
         variable    += object kernel::Variable
                           {
                                    name := '_resp' +
                                    serviceAc.name +
                                    serviceAc.operationsAc.name;
                                    type := self.returnValueType
                           };
         action      += newAction;
         }
}
```

El servicio compuesto final, con la operación especializada `getTripDetailes`, se muestra en la Figura A.4.3.

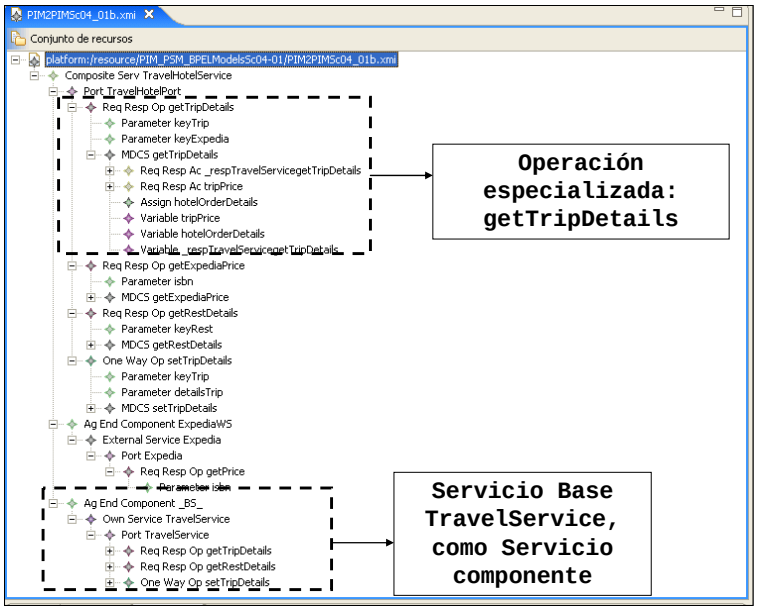


Figura A.4.3 Servicio compuesto final con operaciones especializadas

A.4.3 Últimos pasos: generación de código a partir de la agregación de servicios

Una vez obtenido el servicio compuesto anterior la generación de código termina aplicando los pasos de transformación del escenario tres (**pasos tres** y **cuatro** del escenario). Puesto que ya se ha expuesto un ejemplo de este escenario, no se repite en este apéndice.

APÉNDICE 5

Lenguajes de transformación de modelos

Se han utilizado los lenguajes de transformación de modelos QVT y *MOFS-cript* para la implementación de los escenarios. Este apéndice ofrece una explicación breve de ambos.

A.5.1 QVT: el lenguaje de transformación de modelos

QVT es resultado de una solicitud de propuesta (*Request for proposal-RFP*) de la OMG sobre la necesidad de un lenguaje de transformación de modelos que sea compatible con las tecnologías restantes del MDA (UML, MOF, OCL, etc.).

La especificación de QVT tiene una doble naturaleza (ver Figura A.5.1) : declarativa e imperativa. La parte declarativa se divide a su vez en una arquitectura de dos capas: relacional (*Relations*) y núcleo (*Core*).

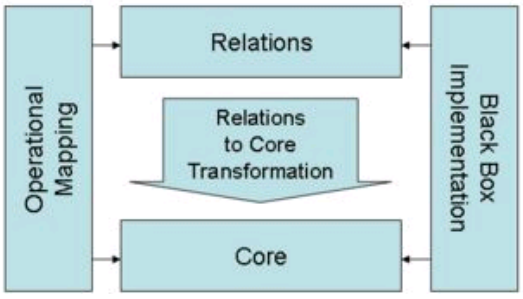


Figura A.5.1 La arquitectura de dos capas del QVT

Cada capa se organiza de la siguiente manera:

- La capa *relacional* posee un metamodelo y un lenguaje que soporta patrones complejos de empate de objetos, así como la creación de los mismos. También tiene la capacidad implícita de crear de crear trazas entre elementos de modelado correspondidos durante las transformaciones.
- La capa *núcleo* posee también un metamodelo y un lenguaje definidos usando un conjunto de extensiones mínimas a EMOF (*Essential MOF*) [57] y OCL. Todas las clases que representan trazas se definen explícitamente como modelos MOF y la creación y eliminación de sus instancias se definen de la misma forma como se define la creación y eliminación de cualquier otro objeto.

A.5.1.1 Las relaciones

Permiten especificar de manera declarativa relaciones entre modelos MOF. Las relaciones pueden también evaluar que otras se cumplan entre elementos particulares de modelado que han sido empatados por los patrones de relación. La semántica de las relaciones se define mediante lenguaje natural y lógica de predicados así como también mediante una transformación estándar para cualquier modelo de relaciones y modelo núcleo con semántica equivalente. Esta transformación puede utilizarse como una semántica formal para la relaciones o como una forma de traducir un modelo de relaciones a un modelo núcleo para ejecución sobre un motor que implemente la semántica del núcleo.

A.5.1.2 El núcleo

Es un pequeño modelo/lenguaje que sólo soporta empatamiento de patrones evaluando condiciones sobre un conjunto de variables contra un conjunto de modelos. El núcleo trata a todos los elementos de modelado -tanto de los modelos fuente, destino y de traza- de manera simétrica. Tiene la misma potencia del lenguaje de relaciones y, por su relativa simplicidad, su semántica se puede definir de manera más simple, aunque la descripción de transformaciones requieren más detalle. A diferencia de las relaciones donde los modelos de traza se pueden deducir de la

descripción de las traducciones, aquí no es posible y deben ser explícitamente definidas. El modelo núcleo se puede implementar directamente o utilizarse como referencia para la semántica de las relaciones, las cuales se corresponden al núcleo.

A.5.1.3 Implementaciones imperativas

Además de los lenguajes de relaciones y núcleo, existen dos mecanismos para invocar implementaciones imperativas de transformaciones desde el lenguaje de relaciones o el núcleo: un lenguaje estándar, de Mapeos Operacionales (*Operational Mappings*); así como uno no estándar, de Operaciones MOF de caja-negra (*Black-box MOF Operation*). Cada relación define una clase la cual será instanciada para trazas entre elementos de modelado que están siendo transformados y que a su vez tiene mapeos uno-a-uno a un operación que implementa un Mapeo Operacional o una Operación MOF de caja-negra.

A.5.1.4 El lenguaje de Mapeo Operacional

Es un lenguaje que provee extensiones OCL con efectos laterales que permiten un estilo más procedural y una sintaxis concreta familiar para los programadores imperativos. Los mapeos operacionales permiten implementar una o más relaciones cuando resulta difícil ofrecer una especificación puramente declarativa sobre como una relación se debe poblar. Una transformación completa puede escribirse en este lenguaje en un estilo imperativo. Una transformación escrita completamente usando Mapeos Operacionales es llamada transformación operacional.

A.5.1.5 Implementaciones de caja negra

Permite la implementación de operaciones que no es posible realizar con los lenguajes anteriores. Es una forma de crear conectores (*plug-ins*) a la infraestructura QVT existente, con lo cual es posible por ejemplo: crear algoritmos complejos que se puedan codificar en cualquier lenguaje de programación, utilizar bibliotecas de dominios específicos (matemáticas, estadísticas, etc.) o facilitar la implementación de algunas partes complicadas de alguna transformación.

A.5.1.6 Escenarios de ejecución

La semántica del lenguaje núcleo (o relacional) permite los siguientes escenarios de ejecución:

- Transformaciones *check-only* para verificar que los modelos están relacionados de alguna manera.
- Transformaciones en un solo sentido.
- Transformaciones bi-direccionales.
- La habilidad de establecer relaciones entre modelos pre-existentes.
- Actualizaciones incrementales (en cualquier dirección) cuando un modelo relacionado ha sido cambiado después de una ejecución inicial.
- La habilidad de crear y eliminar objetos y valores, así como también la capacidad de especificar cuales objetos y valores no deben ser modificados.

Los mapeos operacionales y de caja-negra restringen estos escenarios permitiendo solamente transformaciones en una sola dirección. La transformación bi-direccional solamente es posible si se provee la transformación inversa de manera separada. Las capacidades restantes, sin embargo, están disponibles con las ejecuciones imperativas e híbridas.

En particular la versión QVT Borland Together[56] que ha sido utilizada para la implementación de transformación de modelos de esta tesis utiliza implementa solamente mapeos operacionales. Es una versión, que aunque no posee todas las capacidades de la especificación OMG, ha permitido la implementación de las transformaciones de modelos centrales del presente trabajo de investigación (PIM-PIM, PIM-PSM y PSM-PSM).

A.5.2 MOFScript: el lenguaje de generación de código a partir de modelos

Aunque la herramienta seleccionada (*Borland Together*) posee la capacidad de llevar a cabo transformaciones *Model-A-Texto*, sus posibilidades se basan en el manejo de las clases de implementación Java del metamodelo EMF y el uso particular de las facilidades de manejo de entrada-salida del lenguaje. Revisando las opciones alternativas y buscando una mejor (especialmente enfocada a la generación de texto a partir de modelos MOF) se ha optado por la alternativa *MOFScript*[58] que

constituye una de las primeras respuestas a la solicitud de propuestas de la OMG. Aunque a la fecha no ha sido aceptada completamente como la opción a utilizar en MDA, en la práctica y experiencia que se ha tenido ha resultado suficiente para la implementación de las transformaciones requeridas para esta tesis. A continuación una discusión breve de la herramienta.

A.5.2.1 Aspectos generales de MOFScript

MOFScript representa una respuesta a la solicitud de propuesta (RFP) de la OMG para el soporte del paso de transformación de modelos a representaciones textuales que permitan la generación de artefactos útiles a partir de modelos, tales como código específico de plataforma o documentación.

MOFScript es un subproyecto de Eclipse que busca cubrir los siguientes aspectos en el contexto de generación de texto en ingeniería de software:

- Generación de texto a partir de modelos MOF.
- Ofrecer mecanismos de control, tales como manejo de ciclos y estatutos condicionales.
- Tener capacidad para manejo de *strings*.
- Tener capacidad de salida de expresiones que hacen referencia a elementos de modelado.
- Tener la capacidad de especificar archivos de salida para la generación de texto.
- Trazabilidad entre modelos y texto generado.
- Ofrecer una interfaz fácil de usar.

La implementación actual de *MOFScript* soporta las tareas de análisis sintáctico, verificación, edición y ejecución de código *MOFScript*.

A.5.2.2 Arquitectura MOFScript

La herramienta *MOFScript* está construida con una arquitectura de dos niveles (ver Figura A.5.2): de componentes de la herramienta (*MOFScript Tool Components*) y de componentes de servicios (*MOFScript Service Components*). Los componentes de herramientas son herramientas de usuario final que ofrecen capacidades de edición e interacción con los servicios. Los servicios ofrecen las capacidades de análisis sintáctico, verficación y ejecución del lenguaje de transformación.

El lenguaje se representa mediante un modelo EMF (llamado *modelo MOFScript*) que se construye mediante el analizador sintáctico. Este modelo se utiliza para la verificación semántica y ejecución. La herramienta *MOFScript* se implementa como un conector (*plug-in*) Eclipse utilizando el conector EMF para el manejo de modelos y metamodelos.

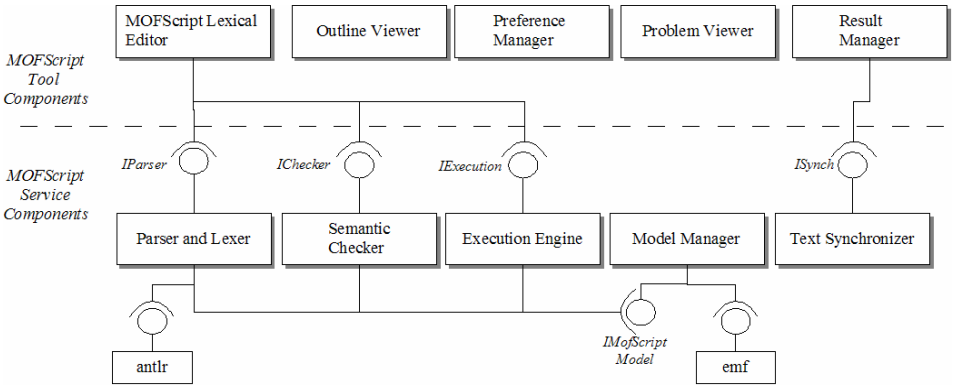


Figura A.5.2 Arquitectura MOFScript

Los componentes de servicios consisten de las siguientes partes: el gestor de modelos (*Model Manager*), un componente basado en EMF que permite la administración de los modelos *MOFScript*. Los analizadores lexicográficos y sintácticos responsables del análisis sintáctico de las definiciones textuales de las transformaciones *MOFScript* y de la construcción del modelo *MOFScript* usando el gestor de modelos. El verificador semático (*Semantic Checker*) provee funcionalidad para la verificación correcta de transformación con respecto a las reglas llamadas, referencias a los elementos del metamodelo, entre otros aspectos. El motor de ejecución (*Execution Engine*) maneja la ejecución de la transformación, interpretando un modelo y produciendo la salida textual. El sincronizador de texto (*Text Synchronizer*) maneja la trazabilidad entre el texto generado y el modelo original, buscando sincronizar el texto en respuesta a cambios en el modelo y viceversa.

MOFScript es desarrollado por una comunidad en SINTEF, soportado y probado por el Proyecto Integrado Europeo MODELWARE (Proyecto IST 511731).

Referencias

[1] Murugesan San, Deshpande Y., Hansen S. and Ginige A. *Web Engineering: a New Discipline for Development of Web-based Systems*. WebEngineering 2000, LNCS 2016, Page(s) 3-13. Springer-Verlag Heidelberg 2001.

[2] S. Pressman Roger. *Can Internet-Based Applications Be Engineered*. IEEE Software, Volume 15, Issue 5. September/October 1998. Page(s): 104-110.

[3] S. Pressman Roger *What a Tangled Web We Weave*. IEEE Software, Volume 17, Issue 1. January/February 2000. Page(s): 18-21.

[4] Schwabe D. and Rossi G. *An Object-Oriented Approach to Web-based Application Design*. Theory and Practice of Object Systems, vol. 4 no. 4, 1998, Page(s): 207-225.

[5] Schmid H. A., Rossi G. *Modeling and Designing Processes in E-Commerce Applications*. IEEE Internet computing. Volume 8, Issue 1. January-February 2004. Page(s): 19-27.

[6] Schmid H.A., Falkenstein F., Rossi G. *Components for the Reuse of Activities in Web Applications*. Proc. 7th Int'l Conf. Object-Oriented Information Systems (OOIS 01), Springer 2001, Page(s): 191-200.

[7] Gómez J., Cachero C., Pastor O. *On Conceptual Modeling of Device-Independent Web Applications: Towards a Web Engineering Approach*. ISSN:1070-986X. IEEE Multimedia 8(2). 2001. Page(s): 26-40.

[8] Koch, Krauss. *The expressive power of UML-based engineering*. Proceedings of the IWWOST'02. CYTED. Page(s): 105-119.

[9] Fons J. Pelechano V., Albert M. and Pastor O. *Development of Web Applications from Web Enhanced Conceptual Schemas*. Proc. of the International Conference on Conceptual Modelling, 22nd Ed. ER'03. Chicago, EEUU, 13-16 october 2003. Page(s): 232-245.

[10] Pastor O., Gómez J., Insfrán E. and Pelechano V. *The OO-Method Approach for Information Modelling: From Object-Oriented Conceptual Modeling to Automated Programming*. Information Systems Elsevier Science. Vol. 26, Page(s): 507-534, Number 7, 2001.

[11] Fons J., Valderas P., Pastor O. *Specialization in Navigational Models.* In: Argentine Conference on Computer Science and Operational Research. Subserie ICWE. Vol. 31, Page(s): 16-31 ISSN: 1666-6526 (2002)

[12] Ceri S., Fraternali P., Bongio A. *Web Modeling Language (WebML): a Modeling Language for Designing Web Sites*. WWW9 Conference, Amsterdam, The Netherlands. Page(s): 137-157. ISSN: 1389-1286. May 2000.

[13] Brambilla M., Ceri S., Comai S., Fraternali P., Manolescu I.: *Model-driven Development of Web Services and Hypertext Applications,* SCI2003, Orlando, Florida, July 2003

[14] W3C. *WSDL. Web Service Description Language*. http://www.w3.org/TR/WSDL

[15] Alonso G., Casati F., Kuno H., Machiraju V. *Web Services. Concepts, Architectures and Applications*. Springer –Verlag Berlin Heidelberg 2004.

[16] G. Bracchi and B. Percini. *The Design Requirements of Office Systems*. ACM Transactions on Office Information Systems, Volume 2, Issue 2. Page(s): 151-170. Apr 1984. ISSN: 1046-8188.

[17] A.D. Birrell and B.J. Nelson. *Implementing remote procedure calls*. ACM Transactions on Computer Systems, Volume 2, Issue 1. Page(s): 39-59, Feb. 1984.

[18] J. Gray and A. Reuter. *Transaction Processing:Concepts and Techniques*. Morgan Kaufmann Publishers, 1993.

[19] Object Management Group. *CORBAservices: Common Object Services Specification*. 1997.

[20] K. Boucher and F. Katz. *Essential guide to object monitors*. John Wiley & Sons, 1999.

[21] IBM. *WebSphere MQ Integrator Broker: Introduction and Planning*. june 2002.

[22] Baina K., Benatallah B., Casati F. and Toumani F. *Model-Driven Web Service Development*. Springer- Verlag. CAiSE '04, 7-11 June 2004, Riga, Latvia.

[23] T. Andrews et al. *Business Process Execution Language for Web Services.* Version 1.1.

[24] ebXML (Electronic Business using eXtensible Markup Language). *ebXML-Enabling a Global Electronic Market*. http://www.ebxml.org. Consultado el día 5 de noviembre de 2007.

[25] UN/CEFACT. *UN/CEFACT Modelling Methodology (UMM).* http://www.unece.org/cefact/umm/umm_index.htm. Consultado el día 5 de noviembre de 2007.

[26] P. Krutchen. *The Rational Unified Process: An introduction*. 3rd Ed. Addison-Wesley. 2004.

[27] EDIFACT. *United Nations for Electronic Data Interchange for Administration, Commerce and Transport* (UN/EDIFACT).

[28] Bernstein Philip A. *Middleware: a model for distributed system services*. Communication of the ACM. Vol. 39, Issue 2 (February 1996). Page(s): 86-98. 1996. ISSN:0001-0782.

[29] W3C. *Extensible Markup Language (XML)*. http://www.w3.org/XML

[30] Casati F, Shan MC. *Models and languages for describing and discovering e-services(tutorial)*. In:SIGMOD Conference, p. 626, Santa Barbara, Calif. USA.

[31] W3C. *Universal Description, Discovery and Integration (UDDI)*. http://uddi.xml.org. Consultado el día 5 de noviembre de 2007.

[32] W3C. *Simple Object Access Protocol (SOAP)*. http://www.w3.org/TR/soap

[33] F. Leymann. *Web Services Flow Language. Version 1.0*. Technical report, International Business Machines Corporation (IBM). May 2001.

[34] Microsoft. *Web Services for Business Process Design* (XLANG). http://xml.coverpages.org/xlang.html. Consultado el día 5 de noviembre de 2007.

[35] T. Andrews et al. *Business Process Execution Language for Web Services.* Version 1.1.

[36] OMG. *Model-driven architecture (MDA)*. http://www.omg.org/mda.

[37] Deshpande Y., Hansen S. *Web Engineering: creating a discipline among disciplines*. IEEE Multimedia. Vol. 8, Issue 2. ISSN: 1070-986X. Page(s): 82-87.

[38] Sun Developer Network. *GlassFish*. http://java.sun.com/javaee/community/glassfish/index.jsp. Consultado el día 5 de noviembre de 2007.

[39] IBM. *IBM Web Services Toolkit- A Showcase for Emerging Web Services Technologies.* http://www-3.ibm.com/software/solutions/webservices/wstk-info.html. Consultado el día 3 de marzo de 2006.

[40] Manolescu I., Brambilla M. Ceri S., Comai S. And Fraternalli P. *Model-Driven Design and Deployment of Service-Enabled Web Applications.* ACM Transactions on Internet Technology. Vol. 5, Number 2. May 2005.

[41] Schwabe D., Rossi G. & Barbosa D.J. *Systematic hypermedia application design with OOHDM*. Proc. ACM Conference on Hypertext. (1996). Page(s): 116-128.

[42] Cachero C., Gómez J., Párraga A. Pastor O. *Extending UML for the migration of Legacy Systems to the Web*. JISBD ’01. Almagro, Spain. Nov. 2001

[43] Kristensen B.B. Osterbye K. *Roles: Conceptual abstraction theory and practical languages issues*. Theory and practice of Object Systems, Volume 2, Issue 3. Page(s):143-160, 1996.

[44] Meyer B. *Object-oriented Software construction*. IEEE Press. 1988.

[45] Albert M., Pelechano V. Fons J. Ruiz M. Pastor O. *Implementing UML association, aggregation and composition. A particular interpretation based on a multidimensional framework. CaiSE* 2003. Page(s): 143-148.

[46] Warmer J., Kleppe A. *The Object Constraint Language*. *Getting your models ready for MDA.* Second Edition. Addison Wesley. 2003.

[47] Amazon on-line store. http://www.amazon.com. Consultado el día 25 de septiembre de 2006.

[48] XMethods. http://www.xmethods.com. Consultado el día 25 de septiembre de 2006.

[49] Laudon Jane, Laudon Kenneth. *Management Information Systems*. 8th. Ed. Pearson, Prentice Hall. 2004.

[50] Parnas David L. *On the criteria to be used in decomposing systems into modules*. Communications of the ACM, Vol. 15, No. 12, Dec 1972. Page(s): 1053-1058.

[51] Larman Craig. *Applying UML and Patterns*. Prentice Hall. 3rd ed. 2004.

[52] OMG. *Unified Modeling Language*. http://www.uml.org. Consultado el día 6 de febrero de 2007.

[53] Apache Web Services Project. *Web Services – Axis*. http://ws.apache.org/axis. Consultado el día 13 de marzo de 2007.

[54] OMG. *MOF QVT Final Adopted Specification*. http://www.omg.org/docs/ptc/05-11-01.pdf. Consultado el día 13 de marzo de 2007.

[55] Eclipse. *Eclipse Modeling Framework (EMF)*. http://www.eclipse.org/modeling/emf/. Consultado el día 14 de marzo de 2007.

[56] Borland Co. *Together Architect 2006 Service Pack 1*. http://www.borland.com. Consultado el día 14 de marzo de 2007.

[57] OMG. *Meta Object Facility (MOF) Core Specification formal/06-01-01*. http://www.omg.org/docs/formal/06-01-01.pdf. Consultado el día 14 de marzo de 2007.

[58] Modelware Project. *MOFscript*. http://www.modelbased.net/mofscript/. Consultado el día 15 de marzo de 2007.

[59] W3C. *XML Path Language (XPath)*. http://www.w3.org/TR/XPath. Consultado el día 22 de marzo de 2007.

[60] Gamma E., Helm R., Jonson R., Vlissides J. *Design Patterns : elements of reusable object-oriented software*. Addison-Wesley Professional Computing Series. 1995.

[61] Bambury, P. *A Taxonomy of Internet Commerce*. First Monday, volume 3, number 10 (October, 1998). Disponible en línea http://firstmonday.org/issues/issue3_10/bambury. Consultado el día 16 de abril de 2007.

[62] Eisenmann, T. *Internet Business Models: Text and Cases*. New York: McGraw-Hill Irwin. 2002.

[63] Rappa Michael. *Business Models on the Web*. Disponible en línea: http://digitalenterprise.org/models/models.html. Consultado el día 16 de abril de 2007.

[64] Papazoglou Mike P. *Service-Oriented Computing: Concepts, Characteristics and Directions*. Fourth International Conference on Web Information Systems Engineering (WISE '03). Dec. 2003.

[65] Van Der Aalst Wilm M. P. *Process-oriented architectures for electronic commerce and interorganizational workflow*. Information Systems. Volume 24, Issue 9 (December 1999). Page(s): 639-671. ISSN:0306-4379.

[66] De Troyer O. *Audience-driven Web Design*. Information Modeling in the New Millennium, Page(s): 442-462 (2001).

[67] Epner M. *Poor Project Management Number-One Problem of Outsourced E-Projects*. Research Briefs. Cutter Consortium. 7 November 2000.

[68] Ginige A., Murugesan S. *Web Engineering: An Introduction*. IEEE Multimedia. Volume 8, Issue 1. January-March 2001. Page(s): 14-18.

[69] Ginige A. *Web Engineering: Managing the Complexity of Web Systems Development*. Proc. of the 14th International Conference on Software Engineering and Knowledge Engineering. Ischia, Italy. Page(s): 721-729. ISBN: 1-58113-556-4.

[70] Garzotto F., Paolini P., Schwabe D. *HDM-a model based approach to hypertext application design*. ACM Transactions on Information Systems (TOIS). Vol. 11, Issue 1 (January 1993). Page(s): 1-26. 1993. ISSN: 1046-8188.

[71] Brambilla, M., Ceri, S., Comai, S. and Fraternali, P. (2006) *A CASE tool for modeling and automatically generating web service-enabled applications*. Int. J. Web Engineering and Technology, Vol. 2, No. 4, Page(s): 354-372

[72] Brambilla, M., Ceri, S., Passamani, M. and Riccio, A. (2004) *Managing Asynchronous Web Service Interactions*, IEEE International Conference on Web Services. ICWS 2004, San Diego, CA, USA. vol. 2004, Page(s): 80-87.

[73] OMG-BPMI. *Business Process Modeling Language 1.0*. http://www.bpmi.org. Consultado el día 5 de noviembre de 2007.

[74] W3C. *XML Schema*. http://www.w3.org/XML/Schema. Consultado el día 26 de abril de 2007.

[75] O'Reilly Network. *What is 2.0. Design Patterns and Business Models for the Next Generation of Software*. 2005. http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html. Consultado el día 26 de abril de 2007.

[76] Phifer, G. *The Fifth Generation of Portals Supports SOA and Process Integration*. Gartner Report. 2006. Gartner: Stanford, CT, USA.

[77] Colombo Massimiliano, Di Nitto Elisabetta, Di Penta Massimiliano, Distante Damiano, Zuccalá Maurilio. *Speaking a Common Language: A Conceptual Model for Describing Service-Oriented Systems*. ICSOC 2005. Page(s): 48-60

[78] W3C. *Web Services Architecture*. W3C Working Group Note 11 February 2004. http://www.w3.org/TR/ws-arch/. Consultado el día 9 de mayo de 2007.

[79] Gronmo Roy, Skogan David, Solheim Ida, Oldevik Jon. *Model-driven Web Services Development*. 2004 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE '04).

[80] Skogan David, Gronmo Roy, Solheim Ida. *Web Service Composition in UML*. 8th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2004).

[81] Provost W., XML.com, *UML for Web Services*. http://www.xml.com/lpt/a/ws/2003/08/05/uml.html. Consultado el día 9 de mayo de 2007.

[82] Google. *Google SOAP Search API*. http://code.google.com/apis/soapsearch. Consultado el día 15 de agosto de 2006.

[83] Google. *Interfaz WSDL Google*. http://api.google.com/GoogleSearch.wsdl. Consultado el día 15 de agosto de 2006.

[84] OMG. *Software Process Engineering Metamodel Specification*. Version 1.1 formal/05-01-06. January 2005.

[85] Cockburn Alistair. *Structuring use cases with goals*. 1997. http://alistair.cockburn.us/index.php/Structuring_use_cases_with_goals. Consultado el día 11 de agosto de 2007.

[86] Quintero R., Pelechano V., Fons J., Pastor O. *Desarrollo de Aplicaciones Web mediante la aplicación de MDA a OOWS*. XII Congreso Internacional de Computación (CIC 2003). Instituto Politécnico Nacional. México, D.F. 13-17 octubre de 2003.

[87] Quintero R., Pelechano V., Fons J., Pastor O. *Desarrollo de Sistemas Basados en Web mediante la aplicación de MDA a OOWS*. 10mo. Congreso Internacional de Investigación en Ciencias Computacionales (CIIC 2003). Oaxtepec, Morelos, México. 22-24 octubre de 2003.

[88] Quintero R., Pelechano V., Pastor O., Fons J. *Aplicación de MDA al Desarrollo de Aplicaciones Web en OOWS*. Page(s): 379-388. Jornadas de Ingeniería de Software y Base de Datos (JISBD), VIII-2003 – November, Alicante(Spain). Ernesto Pimentel, Nieves Brisaboa, Jaime Gómez, 84-668-3836-5, 2003.

[89] Quintero R., Pelechano V., Torres V., Ruiz M., *Una solución a la integración de los métodos de ingeniería Web y las aplicaciones B2B. Un caso de estudio*. Page(s): 301 - 312, Workshop Iberoamericano de Ingenieria de Requisitos y Ambientes Software (IDEAS), Is International, VIII, 2005 - May, Valparaiso (Chile), Hernán Astudillo, 2005.

[90] Torres V., Quintero R., Ruiz M., Pelechano V. *Towards the Integration of Data and Functionality in Web Applications. A Model Driven Approach*. Page(s): 33 - 38, Conference on Advanced Information Systems Engineering Forum (CAiSE Forum), Is International, 17, 2005 - June, Porto (Portugal), Orlando Belo, Johann Eder, Oscar Pastor, João Falcão e Cunha , 972-752-078-2, 2005

[91] Quintero R., Torres V., Ruiz M., Pelechano V. *Modelado de aplicaciones web colaborativas basadas en servicios y dirigidas por procesos de negocio B2B.* Page(s): 123 - 132, Jornadas Científico-Técnicas en Servicios Web (JSWEB), I, 2005 - September, Granada (Spain), Esperanza Marcos, José Alonso, Valeria de Castro, José Carlos del Arco, 84-9732-455-2, 2005.

[92] Quintero R., Torres V., Ruiz M., Pelechano V. *A Conceptual Modeling Approach for the Design of Web Applications based on Services*. ACMSE 2006. ISBN: 1-59593-315-8. Page(s): 464-469. Melbourne, Florida. USA.

[93] Quintero R., Pelechano V. *Conceptual Modeling of Service Composition using Aggregation and Specialization Relationships*. ACMSE 2006. ISBN: 1-59593-315-8. Page(s): 452-457. Melbourne, Florida. USA.

[94] Quintero R., Torres V., Pelechano V. *Model Centric Approach of Web Services Composition*. WEWST/ECOWS 06 (European Conference on Web Services). Zurich, Switzerland. 2006. Book Chapter. Book Series: Whistein Series in Software Agent Technologies and Autonomic Computing. Book: Emerging Web Services Technology. ISBN: 978-3-7643-8447-0 (Print) 978-3-7643-8448-7 (Online). Page(s): 65-81. Editor: Birkhäuser Basel. Springer.

[95] Manolescu I., Brambilla M., Ceri S., Comai S., Fraternali P. *Model-driven design and deployment of service-enabled web applications.* ACM Transactions on Internet Technology (TOIT). Volume 5, Issue 3 (August 2005). Page(s): 439-479. ISSN: 1533-5399. 2005.

[96] Brambilla M., Ceri S., Comai S., Fraternali P., Manolescu I. *Model-driven specification of Web Services Composition and Integration with Data-intensive Web applications*. IEEE Bulletin of Data Engineering. December 2002.

[97] Brambilla M., Guglielmetti G., Tziviskou C. *Asynchronous Web Services Communication Patterns in Business Protocols*. WISE 2005 International Conference. New York. Nov 20-22, 2005.

[98] Schwabe D., Pontes R., Moura I. *OOHDM-Web : An Environment for Implementation of Hypermedia Applications in the WWW*. SigWEB Newsletter, Vol. 8, No. 2. June 1999.

[99] Bernees-Lee T., Hendler J., Lassila O. *The Semantic Web*. Scientific American, 2001.

[100] Ankokelar A., Krotzch M., Tran T., Vrandecic. *The two cultures: mashing up web 2.0 and the semantic Web*. International World Wide Web Conference. WWW '07. Alberta, Canada. ISBN: 978-1-59593-654-7. Page(s): 825-834.

[101] Elrad T., Filman R., Bader A. *Aspect-oriented programming: Introduction*. Communications of the ACM. Volume 44, Issue 10. Page (s): 29-32. ISSN: 0001-0782.

[102] Vallecillo A., Koch N., Cachero C. et al. *MDWEnet: A Practical Approach to Achieving Interoperability of Model-Driven Web Engineering Methods*. Position paper. MDWE2007 Workshop. ICWE 2007. July 16-20, 2007. Como, Italy.

[103] Schauerhuber A., Wimmer M., Schwinger W. Kapsammer E., Retschitzegger W. *Aspect-Oriented Modeling of Ubiquitous Web Applications: The aspect WebML approach*. 5th Workshop on Model-based Development for ComputerBased Systems: Domain-Specific Approaches to Model-Based Development, in conjunction with ECBS, Tucson, AZ, USA. March 2007.

[104] Wimmer M., Schauerhuber A., Schwinger W and Kargl H. *On the Integration of Web Modeling Languages: Preliminary Results and Future Challenges*. MDWE2007 Workshop. ICWE 2007. July 16-20, 2007. Como, Italy.

[105] OSOA. *Open SOA Collaboration*. http://osoa.org. Consultado el día 10 de noviembre de 2007.