Departamento de Lenguajes y Sistemas Informáticos
ETS Ingeniería Informática. Universidad de Sevilla
Avda Reina Mercedes s/n. 41012 Sevilla
Tlf/Fax 954 557 139 **E-mail** lsi@lsi.us.es **Web** www.lsi.us.es

"Separación Avanzada de Conceptos en Entornos Web"

Memoria de Tesis Doctoral

presentada por

Dña. Antonia Mª Reina Quintero

Directores

Dr. D. Miguel Toro Bonilla

Dr. D. Jesús Torres Valderrama

Sevilla, Septiembre 2011

Antonia Mª Reina Quintero
Primera Edición, Septiembre 2011
Dpto. Lenguajes y Sistemas Informáticos
E.T.S. Ingeniería Informática
Universidad de Sevilla
Avda. Reina Mercedes, s/n
41012 Sevilla, España

http://www.lsi.us.es/~reinaqu
reinaqu@lsi.us.es

Clasificación (ACM 1998): D.2.2 Design Tools and Techniques: aspect oriented programming; D.2.10 Design: Methodologies;D.2.13 Reusable Software: Reuse Model; D.2.m Miscellaneous: Rapid Prototyping; D.3.2 Language Classifications: Design Languages, Specialized Application Languages; D.3.4 Processors: Code Generation; H.5.4 Hypertext/Hypermedia: Navigation.

Financiación: Este trabajo ha sido financiado por la Comisión Europea (FEDER) y el Gobierno de España bajo los proyectos CICYT MADEIRA (TIC2000-1673-C06-03), NIDO (TIC2003-00369), Red DSDM (TIN2005-25886-E, TIN2008-00889-E/TIN), Desarrollo de Software Dirigido por Modelos y Separación de Conceptos en la Interacción con el Usuario (TIN2006-04652), QSimTest (TIN2007-67843-C06-03), IntegraWeb (TIN2007-64119, TIN2010-21744-C02-01). Financiación complementaria se ha proporcionado por la Junta de Andalucía bajo el proyecto IntegraWeb (P07-TIC-02602).

Universidad de Sevilla

Reunido el tribunal de evaluación de la memoria de tesis doctoral presentada por Antonia Mª Reina Quintero para la obtención del grado de Doctor por la Universidad de Sevilla, tras votación se acordó otorgar la calificación de _____. A juicio de este tribunal y habiendo obtenido un total de _____ votos de sus miembros, se otorga a la tesis la mención de _____.

_____

Dr. D. Ambrosio Toval Álvarez
Catedrático de Universidad
Universidad de Murcia

_____

Dr. D. Juan Pavón Mestras
Catedrático de Universidad
Universidad Complutense de Madrid

_____

Dra. Dña. Lidia Fuentes Fernández
Catedrática de Universidad
Universidad de Málaga

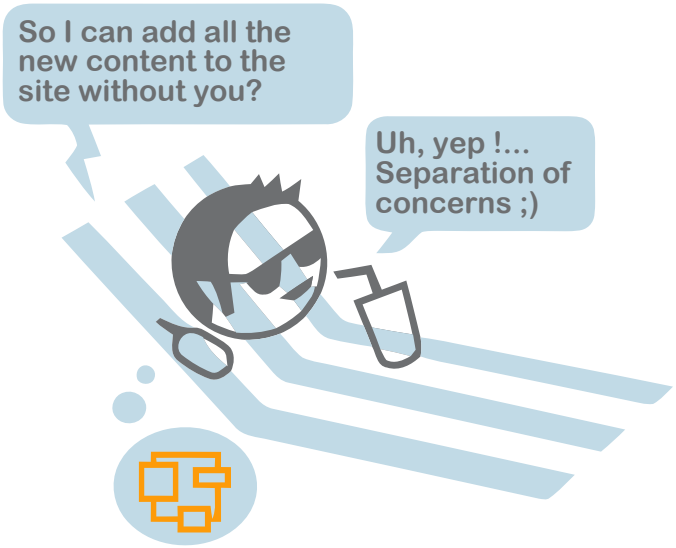
_____

Dr. D. Rafael Corchuelo Gil
Profesor Titular de Universidad
Universidad de Sevilla

_____

Dr. D. Jesús J. García Molina
Catedrático de Escuela Universitaria
Universidad de Murcia

Para que conste a la interesada donde estime oportuno, firmamos la presente acta en Sevilla, a _____.


So I can add all the new content to the site without you?
Uh, yep !... Separation of concerns ;)

A mis padres

A mis abuelos

A la memoria de la memoria

Somos lo que soñamos ser y ese sueño no es tanto una meta como una energía.
Cada día es una crisálida, cada día alumbra una metamorfosis.
Caemos, nos levantamos.
Cada día la vida empieza de nuevo.
La vida es un acto de resistencia y de reexistencia.
Vivimos, revivimos pero todo se sostiene en la memoria.
Somos lo que recordamos, la memoria es nuestro hogar nómada.
Como las plantas o las aves emigrantes los recuerdos tienen la estrategia de la luz.
Van hacia delante a la manera del remero que se desplaza de espaldas para ver mejor.
Hay un dolor parecido al dolor de muelas, a la pérdida física y es perder algún recuerdo que queremos, esas fotos imprescindibles en el álbum de la vida.
Por eso hay una clase de melancolía que no atrapa sino que nutre la libertad y en esa melancolía como espuma en las olas se alzan los sueños.

Manuel Rivas

Índice general

Índice de figuras

Índice de Tablas

Índice de Listados

Agradecimientos

El proceso de realización de una tesis doctoral supone un esfuerzo tan grande, que es difícil de llevar a cabo si lo que vas a defender no es tu causa, si no crees en ella. Se dedican muchas horas de nuestra vida a llevar a cabo un proyecto, a alcanzar un objetivo. Es una tarea difícil, y no todo el mundo está preparado para ello; pero con una gran dosis de trabajo, creatividad y un poco de suerte, llega un día en que se alcanza el punto al que se tiene que llegar, el desenlace. Este desenlace no habría tenido lugar sin la ayuda de muchos que te acompañan durante el camino, por lo que es ahora el momento de agradecerles su compañía.

En primer lugar, me gustaría agradecer a mis directores la libertad que me han dado para realizar este trabajo. A Jesús, por no ponerme límites y animarme a publicar, y a Miguel, por creer en esta tesis, muchas veces, incluso más que yo misma.

Mención especial quisiera hacer a Rafael Corchuelo, mi tercer director, si fuera posible, por todo el tiempo dedicado a este trabajo, sus consejos, sus revisiones y sus críticas. Sin su ayuda, esta tesis no habría salido adelante. Gracias también por todo aprendido, y todo lo que queda por aprender al trabajar a su lado.

Sin el apoyo de los compañeros con los que compartes asignatura, sería prácticamente imposible asistir a distintas actividades relacionadas con la investigación. Me gustaría agradecerles a todos aquellos con los que he compartido las distintas asignaturas por las que he pasado las facilidades para realizar estas labores. Especialmente, quería agradecerle a Rafa Ceballos, la sustitución en IIDSI; a Pepe Riquelme, por escucharme durante estos años; a Juan A. Álvarez, David Ruiz y José A. Pérez, por los momentos compartidos con los sistemas operativos, y a Manolo Rovayo, por todo el apoyo recibido en los momentos difíciles. Gracias también a Manolo Mejías e Isabel Ramos, por facilitarme en todo lo posible los trámites en unas circunstancias difíciles.

Nuevos caminos se abren, y me gustaría agradecer a Rafael Z. Frantz, Sergio Pozo y Mayte Gómez, el que contaran conmigo para trabajar con ellos. Espero que estas colaboraciones sigan dando sus frutos.

Finalmente, gracias a Mayte, Octavio y Rafa por estar ahí cuando los necesitaba y por tantas comidas, cafés y peripecias compartidas. A lo largo de estos años, se han convertido en verdaderos amigos.

Mis amigas, también han participado en esta tesis. Gracias a Cristina, por ofrecerse a leerla antes de la publicación, y a Reyes, que aunque es del gremio, no se ha cansado de escucharme. A las viajeras, Maribel y Sonia, gracias por escucharme en nuestros

viajes. Finalmente, a mis amigas de todas la vida, Nuria, por sus ánimos y su apoyo, sobre todo en el último tramo, y Coral, por haber estado siempre ahí.

Pero, sin lugar a dudas, el apoyo más importante durante este trabajo, especialmente en estos difíciles últimos momentos, ha sido mi familia. A mis padres, por su apoyo incondicional, y creer siempre en mi. A mi abuela Joaquina, por enseñarme lo que es el trabajo. Y a mi hermano, por su apoyo y sus ideas con la portada.

Finalmente, a todos aquellos que durante el camino preguntaban que cuánto faltaba para la meta, gracias por interesaros, ahora ya veo el cartel de llegada.

Abstract

In a deeply interconnected and globalised world, one of the main challenges that the current software development industry has to face is to adapt to changes as inexpensively and fast as possible. This challenge is even more important in Web applications, because the rate of changes is much greater than in traditional software applications. The main reasons for software modifications are related to changes in business processes, changes in user requirements, and the evolution of technological environments.

In the field of Web applications, one of the concerns that changes more frequently is the user interface and, as a consequence, navigation, which is intimately related to the user interface. Not only is navigation a key feature regarding the usability of web sites, but also can have an important impact on the costs of a project. In this context, our hypothesis is that as changes in web navigation are frequent, software companies are very interested in approaches to reduce the impact of these changes. Thus, a design of web navigation that reduces the impact of technology changes and requirements can be a key factor in the development of such applications.

Unfortunately, navigation is a complex domain whose design is frequently specified as a view of conceptual models. However, navigation can be decomposed into different navigational concerns, some of which can be specified completely independently from the conceptual model, e. g., structural, utility navigation or navigation flows. The thesis we defend in this document is that it is desirable to devise a framework that let us apply the advanced separation of concerns principle to navigation design, and help us separate concerns horizontally (separating different navigational concerns) and vertically (separating the concerns that depend on specific technologies from the ones that are technology-agnostic).

In this dissertation, we motivate why this framework is necessary; we present a survey of the state-of-the-art approaches in the research areas closer to our work (Aspect-Oriented Software Development, Model-Driven Software Development, Model-Driven Architecture, and Web Engineering); and, finally, we introduce `MWACSL`, our approach for advanced separation of concerns regarding navigation.

`MWACSL` is an aspect-oriented, model-driven approach for web development, which defines a model architecture inspired on `MDA` to separate concerns vertically, and uses domain specific languages (or `DSL`s) to separate concerns horizontally. These `DSL`s are known as Concern Specific Languages (`CSL`s).

In this document, two different `CSL`s are defined, one of them aims at specifying

structural and utility navigation, and the other one aims at dealing with navigation flows. These CSLs help us separate concerns at the PIM level. This document also introduces a set of model-to-model and model-to-text transformations, and a set of weaving models that allow us to obtain an implementation of these concerns for different platforms, including Spring Web Flow, HTML and JSP. Our approach helps modularise navigation better, which definitely helps us reduce the impact of changes in navigation.

Resumen

En un mundo profundamente interconectado y globalizado, uno de los principales retos a los que se tiene que enfrentar la industria de desarrollo software de hoy en día es conseguir adaptarse a los cambios de la forma menos costosa posible. Este reto es aún más fuerte en las aplicaciones web, ya que el grado y frecuencia de cambios es incluso mayor que en las aplicaciones software tradicionales. Los principales motivos para modificar el software están relacionadas con los cambios en los procesos de negocio, los cambios de los requisitos de los clientes y la evolución de los entornos tecnológicos.

En el ámbito de las aplicaciones web, uno de los aspectos que sufre cambios más frecuentemente es la interfaz de usuario y, como consecuencia, la navegación, íntimamente ligada a las interfaces web. La navegación no sólo es una característica determinante en la usabilidad de los sitios, sino que puede llegar a tener un impacto importante en los costes de un proyecto. En este contexto, nuestra hipótesis de partida es que como los cambios en la navegación de las aplicaciones web son frecuentes, las empresas software tienen mucho interés en propuestas que permitan reducir el impacto de estos cambios. Así, un diseño de la navegación de las aplicaciones web que reduzca el impacto de los cambios de tecnología y de requisitos puede ser un aspecto clave en el desarrollo de este tipo de aplicaciones.

Por desgracia, la navegación es un dominio complejo cuyo diseño frecuentemente se realiza en base al modelo conceptual. Sin embargo, la navegación se puede descomponer en diferentes *concerns*, algunos de los cuales, como la navegación estructural y de utilidad o los flujos de navegación, se pueden definir de forma totalmente independiente al modelo conceptual. La tesis que defendemos en esta memoria es que es conveniente desarrollar un *framework* que permita aplicar al diseño de la navegación una separación avanzada de conceptos, tanto a nivel horizontal (separando diferentes *concerns* navegacionales) como vertical (separando aquellas cuestiones que dependen de tecnologías concretas de las que son independientes de las misma).

En esta tesis se motiva por qué es conveniente este *framework*, se hace un estudio del estado del arte de las propuestas relacionadas con las distintas áreas de investigación con las que está relacionada este trabajo (el Desarrollo de Software Orientado a Aspectos, el Desarrollo de Software Dirigido por Modelos y `MDA`, y la Ingeniería Web), y se presenta `MWACSL`, que es nuestra aproximación para conseguir esta separación avanzada de conceptos en el tratamiento de la navegación.

`MWACSL` es una aproximación orientada a aspectos y dirigida por modelos para el

desarrollo web, que define una arquitectura de modelos inspirada en MDA para conseguir una separación vertical de los conceptos dependientes de plataformas concretas de aquéllos que son independientes de las mismas y que separa los conceptos horizontalmente mediante lenguajes específicos de dominio (o DSL's). A estos lenguajes específicos de dominio que se usan para especificar un *concern* se les denomina CSL.

En esta memoria se especifican dos CSL's para tratar la navegación estructural y de utilidad y los flujos de navegación. Estos CSL's especifican y separan estos *concerns* a nivel independiente de plataforma. En esta memoria también se presentan un conjunto de transformaciones de modelo a modelo, de modelo a texto y una serie de modelos de *weaving* que permiten obtener una implementación de estos *concerns* en distintas plataformas, como Spring Web Flow, HTML o JSP. Con todo esto se consigue una mejor modularización de la navegación, lo que ayuda a conseguir nuestro objetivo inicial, reducir el impacto de los cambios en la navegación.

Parte I

Prefacio

It does not matter how slowly you go so long as you do not stop.

Confucius

1 Introducción

En este capítulo se dibuja el contexto en el que se enmarca esta tesis. En la sección 1.1, se contextualiza el trabajo en las áreas de investigación en las que se encuadra. La sección 1.2 define la hipótesis de trabajo de partida y la tesis a demostrar, mientras que la sección 1.3 resume las contribuciones realizadas para solucionar los problemas detectados en la sección anterior. Luego, la sección 1.4 presenta las contribuciones obtenidas en forma de publicaciones, enmarcando el trabajo dentro de los proyectos de investigación en los que se ha desarrollado la tesis. Por último, la sección 1.5 da una visión de cómo se ha organizado este documento.

1.1 Contexto de la investigación

Uno de los principales problemas a los que se tiene que enfrentar la industria de desarrollo software de hoy en día es estar preparados para afrontar los constantes cambios a los que se ve sometida y, además, hacerlo de la forma menos costosa posible. Los principales focos de cambio en relación al software vienen provocados, por una parte, por la variedad y variabilidad de los requisitos que imponen los clientes a la hora de contratar el desarrollo de cualquier proyecto software y, por otra, por la rapidez con la que evoluciona el entorno tecnológico. Estos cambios se acentúan más en las aplicaciones web, en las que el grado y frecuencia de cambio es mucho más alto que en las aplicaciones software tradicionales [269]. Además, en el contexto de las aplicaciones web, uno de los aspectos más distintivos es la navegación, que no es sólo uno de los aspectos que sufre de cambios frecuentes, sino que es una característica determinante

en la usabilidad de los sitios web [33], pudiendo llegar a tener un impacto importante en los costes de un proyecto [206].

Recientemente han surgido tres disciplinas, el Desarrollo de Software Dirigido por Modelos (DSDM), el Desarrollo de Software Orientado a Aspectos (DSOA) y la Ingeniería Web (IW), que intentan, desde diferentes perspectivas, resolver algunas de las cuestiones relacionadas con este problema. En esta tesis se aboga por la combinación de estas disciplinas. En las siguientes secciones se desgrana cómo encajan las mismas en el contexto de la tesis.

1.1.1 Desarrollo de Software Dirigido por Modelos

El DSDM es una disciplina que tiene como objetivo elevar el nivel de abstracción en el que los desarrolladores crean software. Además, es una disciplina prometedora para tratar con la complejidad de las plataformas [353], que ha convertido a los modelos en el centro del proceso de desarrollo, y que ha ayudado a simplificar y formalizar las diferentes actividades y tareas que forman parte de cualquier ciclo de vida de desarrollo software [169]. Dentro del DSDM existen visiones distintas como Software Factories (SF) [151], Model Driven Architecture (MDA) [282], Model-Integrated Computing (MIC) [196] o los Domain Specific Languages (DSL's).

En esta tesis se combinan dos de estas visiones del DSDM, MDA y DSL's. Los DSL's se utilizan para separar *concerns* a nivel independiente de plataforma, mientras que se adopta la división en niveles de modelado propuesta en MDA para separar aquellas cuestiones que dependen de plataformas concretas de las que son independientes de la misma. A continuación se dan más detalles de estas dos visiones.

Visión del DSDM con MDA Model Driven Architecture (MDA) [282] es la visión del DSDM de la OMG, y tiene como principal objetivo el desacoplar el modo en el que se definen las aplicaciones de la tecnología en la que se ejecutan las mismas. Para conseguir este objetivo, MDA separa la lógica subyacente en una especificación de las propiedades particulares de los *middleware* en los que se va a desplegar mediante una arquitectura con diferentes niveles de modelado: Modelo Independiente de Computación (CIM), Modelo Independiente de Plataforma (PIM) y Modelo Específico de Plataforma (PSM). La OMG recomienda el uso de UML como lenguaje de modelado, y la creación de perfiles como forma de extender o personalizar este lenguaje de modelado, aunque también da la opción de crear nuevos lenguajes de modelado desde cero usando MOF.

Visión del DSDM con DSL's El concepto de Lenguaje Específico de Dominio (o DSL) no es nuevo. Según Fowler [128], un DSL es un lenguaje de programación de expresividad limitada y centrado en un dominio particular. Actualmente, esta definición no se reduce a lenguajes de programación, sino también a lenguajes de modelado. Además, según Kurtev et al. [229], un DSL se define como un conjunto de modelos coordinados.

Los DSL's facilitan la gestión de la variabilidad [180]. Además, como proporcionan abstracciones y sintaxis especializadas para un dominio concreto, permiten evitar el código repetitivo y detalles de implementación de bajo nivel [394]. La sintaxis de un DSL puede ser gráfica, textual o una mezcla de ambas. Además, el construir DSL's usando un *framework* de construcción de DSL's en el marco de la ingeniería de modelos, permite resolver problemas complejos y de evolución de forma más fácil que con las tecnologías de programación orientada a objetos tradicionales [229].

La visión del DSDM mediante DSL's se basa en usar soluciones basadas en modelos para definir la sintaxis y la semántica de un DSL [229], de forma que se use un metamodelo de definición del dominio para definir la sintaxis abstracta, un modelo de transformación que relaciona esta sintaxis abstracta con una de las posibles sintaxis concretas, y otro modelo de transformación que relacione un DSL con otro lenguaje para definir su semántica de ejecución.

Transformaciones de modelos vs. composición de modelos Las relaciones entre modelos se pueden definir mediante transformaciones o mediante modelos de composición. Ambas operaciones tienen como objetivo incorporar nuevas características a un modelo. La principal diferencia entre ambas operaciones es que en la composición de modelos, las nuevas características a incorporar se hacen explícitas, ya que están descritas en uno o más modelos fuentes, mientras que en la transformación de modelos estas características quedan implícitas en las transformaciones.

Aunque el término transformación de modelos es reconocido por todos los investigadores, la composición de modelos es una operación que se ha utilizado en distintas áreas de conocimiento, lo que ha provocado que no exista un vocabulario común y que sea difícil comparar la gran variedad de propuestas que hay en la bibliografía.

1.1.2 Desarrollo de Software Orientado a Aspectos

El Desarrollo de Software Orientado a Aspectos (DSOA) [119] es una disciplina que tiene como objetivo aislar y modularizar los distintos *crosscutting concerns* o aspectos que componen un sistema, componiéndolos posteriormente de forma no invasiva. La capacidad de modularización adicional introducida por los aspectos da como resultado aplicaciones que son más fáciles de mantener, extender y reutilizar [359]. Como consecuencia de esta mejor modularización los conceptos están mejor localizados, de forma que podemos afirmar que es más fácil adaptarse a la variabilidad de los requisitos de los clientes, puesto que es más fácil determinar dónde están los puntos del sistema afectados por los cambios. Aunque el desarrollo de software orientado a aspectos surge a nivel de programación intentando modularizar algunos conceptos que se diseminan por todo el programa y que no se recogen bien en los lenguajes de programación tradicionales, pronto se convirtió en una filosofía que se extendió a todo el ciclo de vida.

Una de las formas de separar los *crosscutting concern* es creando nuevos lenguajes o formalismos que permitan representar estos conceptos. Estos lenguajes puede ser de

propósito general, es decir, que permitan representar cualquier *crosscutting concern* o específicos para un dominio concreto, en el que se representan conceptos relacionados con un *crosscutting concern*, como la sincronización.

En esta tesis se usan lenguajes específicos de dominio para separar *concerns* a nivel independiente de plataforma, así que en el siguiente apartado se darán algunos detalles más de esta estrategia.

DSAL's En el contexto de la orientación a aspectos, los `DSL's` forman parte del germen de los trabajos primitivos de este área, así, por ejemplo, en la tesis de Lopes [237] se especificaban dos lenguajes específicos de dominio llamados `COOL` y `RIDL` para tratar con los aspectos de sincronización y serialización, respectivamente. Un lenguaje de aspectos específico de dominio [62] se puede definir como un `DSL` para expresar *crosscutting concerns*, o más formalmente, un `DSL` cuyos programas se componen de forma no funcional con otros programas.

Por otra parte, los dominios pueden estar compuestos de varios *concerns*. Los lenguajes específicos de *concerns* [62] (`CSL`'s) son lenguajes que se usan para un *concern* específico, y no para un dominio. La principal diferencia entre un `CSL` y un `DSAL` es que los `CSL`'s también abarcan los *concerns* que no son *crosscutting* y que se pueden componer usando técnicas de composición funcionales.

Uno de los problemas que presentan estos lenguajes es que, en la práctica, cada uno requiere su propio *weaver*, con lo que es difícil combinarlos, ya que cada lenguaje tiene su propia semántica.

En esta tesis se trabaja con `CSL`'s en el espacio técnológico de los modelos (o *modelware*), a diferencia de los primeros *DSAL*'s que se definían en el espacio tecnológico de las gramáticas (o *grammarware*). Según [228], un espacio tecnológico es un contexto de trabajo, con un cuerpo de conocimiento, herramientas, habilidades requeridas. Inicialmente, los autores consideran cinco espacios tecnológicos: los lenguajes de programación, la ingeniería de ontologías, los lenguajes basados en `XML`, los sistemas de gestión de bases de datos y `MDA`. El espacio tecnológico de los lenguajes de programación es a lo que se ha llamado después *grammarware* y a `MDA`, *modelware*.

1.1.3 Ingeniería Web

La red de redes ha supuesto una revolución, haciendo que el volumen de negocios en la misma sea cada vez mayor. A la vez que ha aumentado el número de usuarios de Internet, se ha incrementado también el número de empresas que ofrecen sus productos y servicios a través de la red. Este boom de las comunicaciones no solo ha provocado un incremento del número de aplicaciones web, sino también de su complejidad. Mientras que entre 1995 y 1998 la mayoría de los sitios web estaban compuestos principalmente por páginas `HTML` estáticas, desde 1998 el número de sitios construidos solamente con páginas estáticas ha ido decrementándose, dando paso a aplicaciones de servidor complejas, que utilizan la tecnología web para desarrollar su interfaz de usuario. Así,

actualmente las aplicaciones web están formadas por diversos componentes como archivos `HTML` planos, una mezcla de archivos `HTML` y programas, lenguajes de *script*, bases de datos, imágenes e interfaces de usuario complejas [280]. De esta forma, las aplicaciones web se han convertido en sistemas tan sofisticados como muchas aplicaciones cliente-servidor con interfaz de usuario.

En la bibliografía se pueden encontrar varias definiciones del término aplicación web. En general, se utiliza para distinguir a ciertos tipos de sitios web. En esta tesis una aplicación web se considera como una aplicación de servidor que utiliza tecnología web para desarrollar su interfaz de usuario, puesto que muchas de las aplicaciones web modernas tienen que ejecutar transacciones conversacionales con el cliente, requiriendo que el servidor conozca tanto el estado del cliente como los límites de la transacción.

En este contexto surge la Ingeniería Web, que toma prestados muchos principios y conceptos de la ingeniería del software, pero aplicándolos a la problemática propia de aplicaciones web y se centra en la adopción de modelos, metodologías sistemáticas, y herramientas para el diseño, desarrollo y evaluación de aplicaciones web de alta calidad [55].

El trabajo desarrollado en esta tesis se enmarca dentro del contexto de la ingeniería web puesto que se centra en el desarrollo de aplicaciones web y especialmente en la navegación.

1.1.4 Navegación

La navegación en una aplicación web juega un papel fundamental a la hora de dar forma a las experiencias de los usuarios con la Web [206]. La noción intuitiva de navegación está relacionada con el hecho de pulsar un enlace y moverse de una página web a otra. Sin embargo, su definición es algo más sutil [298], y cabe preguntarse si cada vez que pulsamos un enlace en una aplicación web estamos navegando. Según [354] la navegación es el proceso cognitivo de adquirir conocimiento sobre un espacio, estrategias para moverse a través del espacio, y cambiar el metaconocimiento del espacio. Desde esta perspectiva, los autores distinguen entre operaciones de interfaz y operaciones de navegación [354]. Así, por ejemplo, los enlaces que aparecen en un motor de búsqueda como *Google* para paginar los resultados no son concebidos como navegación, sino como un mecanismo de *scroll* que forma parte de la interfaz de usuario. Sin embargo, hay otros autores como [206], que consideran que este tipo de enlaces son un mecanismo de navegación al que denominan navegación de paginación.

El diseño de la navegación es una tarea compleja que debe coordinar los objetivos de los usuarios con los objetivos del negocio. Además de la Ingeniería Web, otra disciplina relacionada con el diseño de la navegación es la Arquitectura de la Información [413]. Mientras que la arquitectura de la información está relacionada con organizar, estructurar y etiquetar la información mientras que la ingeniería web tiene más que ver con el desarrollo técnico y visual. Uno de los problemas actuales es que estas dos disciplinas suelen estar desconectadas [22], dando como resultado malos diseños porque la tecno-

logía de la información está demasiado separada o mal integrada dentro del proceso de diseño.

Dentro de MWACSL, nos hemos centrado en la navegación, un dominio complejo que está compuesto por distintos tipos de *concerns* [197]. Así se definen lenguajes de modelado para dos *concerns* navegacionales, la navegación estructural y de utilidad y los flujos de navegación. Además, la desconexión entre la Ingeniería Web y la Arquitectura de la Información se ha abordado teniendo en cuenta ambas perspectivas, de forma que el DSL diseñado para tratar con la navegación estructural y de utilidad se ha inspirado en los mapas de sitio web, que son unos entregables bastante utilizados en el área de la arquitectura de la información. Además, en el capítulo 5, que se centra en el modelado de la navegación, se tienen en cuenta ambas perspectivas.

1.2 Hipótesis de trabajo y tesis

La industria del desarrollo software tiene que prepararse para afrontar los cambios tanto de requisitos como de tecnología de la forma menos costosa posible. El grado y frecuencia de cambios es más alto en las aplicaciones web que en las aplicaciones software tradicionales [269]. Según Murugesan y Ginige [270], la gestión de la evolución y el cambio en las aplicaciones web es un reto técnico, de organización y de gestión mucho más exigente que en las aplicaciones web tradicionales, incluyendo dentro de estos retos importantes la proliferación de nuevas tecnologías y estándares [269], que presionan a las empresas para que las adopten si quieren seguir siendo competitivas. Dentro del propio contexto de las aplicaciones web, la navegación es una de las características críticas [54] que sufre más cambios y una de las características determinantes en la usabilidad del sitio [33, 113], pudiendo llegar a tener un impacto importante en los costes de un proyecto [206].

El desarrollo de aplicaciones web puede mejorar si conseguimos que el diseño de la navegación de las aplicaciones web se pueda realizar de forma que se adapte mejor a los cambios de tecnología y de requisitos. Como la navegación es un dominio complejo, se podrá adaptar mejor a los cambios de requisitos si se separan los distintos *concerns* que la componen y si estos *concerns* se describen mediante lenguajes de modelado que no tengan detalles de las plataformas en la que se implementan, lograremos también una mejor adaptación a los cambios de tecnología.

Para conseguir la separación de los detalles de las plataformas concretas se utiliza una arquitectura de modelos inspirada en MDA. La separación de *concerns* se consigue gracias a la definición de un DSL especificado en el espacio tecnológico de los modelos por cada uno de los *concerns* a separar. La relación entre *concerns* se define mediante *weaving* de modelos y la adaptación a las plataformas concretas se consigue mediante transformaciones de modelos.

Como conclusión, nuestra hipótesis de trabajo es la siguiente:

Los cambios en la navegación de las aplicaciones web son frecuentes y,

por lo tanto, las empresas de desarrollo software tienen mucho interés en propuestas que permitan reducir el impacto de estos cambios.

Teniendo en cuenta todo esto, como tesis sostenemos que:

Es posible desarrollar una propuesta basada en modelos específicos de dominio a nivel independiente de plataforma que separe los distintos concerns *que componen la navegación de una aplicación web.*

1.3 Resumen de contribuciones

Las contribuciones de esta investigación son las siguientes:

1. La concepción de la navegación desde la perspectiva de la orientación a aspectos.
2. El considerar la ingeniería de modelos como una forma de implementar lenguajes específicos de *concerns*, especificando los lenguajes específicos de *concerns* en el espacio tecnológico de los modelos.
3. La definición de un lenguaje de modelado específico de dominio para tratar y especificar la navegación estructural y de utilidad (sección 8.2) junto con una herramienta de modelado (sección 12.2.1).
4. La definición de un lenguaje de modelado específico de dominio para especificar la navegación controlada o flujos de navegación (sección 8.3).
5. La definición de un lenguaje de modelado específico de dominio para especificar flujos de navegación controlada en Spring Web Flow [89] (sección 9.3).
6. La definición una serie de transformaciones usadas como prueba de concepto para generar la navegación estructural y de utilidad en `HTML` (sección 8.2.3).
7. La definición de una serie de modelos de *weaving* para combinar, por una parte, navegación e interfaz, y por otra, a los propios *concerns* navegacionales (secciones 10.4.2 y 10.4.3, respectivamente).

1.4 Publicaciones

La sección de publicaciones se ha dividido en cuatro grandes bloques. En primer lugar, se enumeran los proyectos de investigación en los que se ha enmarcado este trabajo de tesis. Luego se hace un resumen de las publicaciones relacionadas directamente con esta tesis y en las que la candidata figura como primera autora. Después se resumen otras publicaciones relacionadas de forma más tangencial con esta tesis, y que son fruto de la colaboración con otros autores. Por último se presenta un resumen y una clasificación de las citas que se han realizado a los artículos directamente relacionados con la tesis.

Proyectos de Investigación El trabajo presentado en esta tesis ha sido financiado por los siguientes proyectos, ordenados descendentemente por fecha:

1. Integración de Aplicaciones y Datos en la Web (TIN2010-21744-C02-01).
2. Red de Desarrollo de Software Dirigido por Modelos (TIN2008-00889-E/TIN).
3. Metodología y Herramientas para la Integración de Islas de Datos Amigables en la Web (P07-TIC-02602, TIN2007-64119).
4. Calidad predecible y gestionada mediante simulación y técnicas de pruebas en etapas tempranas. (QSimTest) (TIN2007-67843-C06-03).
5. Desarrollo de software dirigido por modelos y separación de conceptos en la interacción con el usuario (TIN2006-04652).
6. Red de Desarrollo de Software Dirigido por Modelos (Red DSDM) (TIN2005-25886-E).
7. Navegación e interacción con el usuario en el desarrollo de sistemas de información Web: métodos, técnicas y herramientas (TIC2003-00369).
8. MADEIRA (Metodologías y Arquitecturas para la Difusión de Información por Internet) dentro del Proyecto Coordinado DOLMEN (Distributed Object, Languages, Methods And Environments)(TIC2000-1673-C06-03).

Resumen de publicaciones Durante el desarrollo de esta tesis, la autora ha publicado los resultados de la misma en distintos foros. Un listado detallado de estas publicaciones se puede ver en la sección A.1 del Apéndice A, y la versión más actualizada del listado se puede encontrar en el sitio web de la autora (`http://www.lsi.us.es/~reinaqu/mispapers.html`).

En este apartado solamente se hace referencia a las publicaciones más relevantes, bien por el sitio en el que se han publicado, bien por las citas que han obtenido y en las que la autora de esta tesis figura como primer autor de la publicación. Las publicaciones más relevantes por las referencias que se han recibido de otros autores se muestran en la tabla 1.1, en la que aparecen ordenadas por número de citas. Entre estas publicaciones citadas cabe destacar las siguientes:

- [LSI-TR'00] [317] es una de nuestras primeras publicaciones, un informe técnico que aunque no es una publicación muy relevante por el sitio en el que está publicado, sí merece ser tenida en cuenta en este resumen por el número de referencias de otros autores. En el informe técnico se hacía una primera exploración del estado del arte de una de las áreas en las que se enmarca este trabajo de tesis, la programación orientada a aspectos. Como consecuencia de su carácter introductorio, ha servido como bibliografía a varios estudiantes de doctorado, sobre todo de habla

Ref.	Acron.	Publicación	Num. Citas
[332]	UML-AOM'04	Developing Generic Solutions with Aspects	25
[317]	LSI-TR'00	Visión General de la Programación Orientada a Aspectos	17
[331]	ECOOP-AAOS'03	Aspect-Oriented Web Development vs. Non Aspect-Oriented Web Development	12
[327]	ENTCS'07	Using aspect-oriented techniques to improve the reuse of metamodels	7
[326]	ECOOP-MAHCC'05	Weaving AspectJ Aspects by means of Transformations	7
[322]	GAOP'02	Analysing the Navigational Aspect	6
[323]	DDMA'02	Separating the Navigational Aspect	5
[340]	JISBD-DSOA'03	Una Experiencia Práctica Reutilizando Aspectos	3
[324]	RCC'04	Components+Aspects: A General Overview	3
[333]	JISBD-DSOA'06	Hacia Lenguajes de Metamodelado Orientados a Aspectos	1
[335]	ICWE-AEWSE'07	Improving the Adaptation of Web Applications to Different Versions of Software with MDA	1

Tabla 1.1: Relación de publicaciones citadas

hispana. En este sentido, este informe técnico también ha aparecido como bibliografía recomendada en la asignatura *Programación Dirigida a Objetos (PDO)*[1] impartida por el Dpto. de Lenguajes y Sistemas Informáticos de la Universidad de Granada, en el *Seminario de Sistemas de Información II* dentro de la carrera profesional de Computación e Informática, impartido en Perú, y en el capítulo 1 del libro *Programación Orientada a Objetos con Java*[2] [166].

- [GAOP'02] [322] y [DDMA'02] [323] son dos publicaciones en las que se introduce por primera vez una visión de la navegación en el contexto de las aplicaciones web desde una perspectiva orientada a aspectos, considerándola como un *crosscutting concern*. Hasta donde nosotros conocemos, éste fue el primer trabajo en el que se muestra una concepción de la navegación desde esta perspectiva, concepción que después han seguido otros autores como [197, 170]. Además de encontrarse ambos entre nuestros trabajos citados, [GAOP'02] se recomienda en el sitio web de Pressman & Associates llamado Downloadable Reference Library como bibliografía complementaria al capítulo 18 del libro de Pressman, titulado *Analysing modelling for web applications*[3] .

[1] `http://tinyurl.com/3arzxre`
[2] `http://dgp.lcc.uma.es/contenidos/erratas.html`
[3] `http://www.rspa.com/reflib/AnalysisWebApps.html`

- [ECOOP-AAOS'03] [331] aparece la tercera en el ranking de nuestras publicaciones más citadas, y también está en la línea de introducir la filosofía del desarrollo de software orientado a aspectos en el desarrollo de aplicaciones web, comparando para ello el desarrollo de una aplicación web realizada con dos tecnologías distintas, la orientación a aspectos y otra tecnología web basada en componentes y XML.

- [UML-AOM'04] [332] es nuestra publicación más citada y recoge dos aportaciones interesantes. Por una parte, se hace una revisión de las propuestas de modelado orientadas a aspectos que había en el momento de su publicación, y por otra, es donde se sientan las bases de lo que posteriormente será MWACSL, proponiendo la arquitectura de modelos en niveles inspirada en MDA.

- [ECOOP-MAHCC'05] [326] es otra publicación citada y en ella se combinan los mecanismos de DSOA y DSDM para trabajar con modelos a nivel específico de plataforma, siendo citada, entre otros, en un artículo de la biblioteca de recursos de Microsoft MSDN Library[4]

- [ENTCS'07] [327] es otra publicación citada y en ella se propone usar los mecanismos definidos en el DSOA para hacer que los metamodelos sean más reusables. Además, esta publicación se ha utilizado como bibliografía complementaria en el curso *Tecnologies de l' Internet - Ingénierie Dirigée par les Modèles*[5] , impartido en el *Master Informatique Technologies de l' Internet* de la *Université de Pau et des Pays de l' Adour*, Francia.

Finalmente, entre las publicaciones más recientes, aunque aún no han recibido citas de otros autores, hay que destacar:

- [IJCAT'08] [319] es una publicación en la revista *International Journal of Computer Application in Technology* en la que se hace una revisión y comparativa de aquellas propuestas de la bibliografía que modelan la navegación y aplican tanto ideas del DSOA como del DSDM.

- [WEBIST'10] [329] es una de las publicaciones más recientes y en ella se describe una primera versión del metamodelo para tratar con la navegación estructural y de utilidad.

Resumen de publicaciones relacionadas En este apartado se destacan aquellas publicaciones relevantes que, aunque no forman parte directamente de MWACSL, han resultado de la colaboración del la autora de esta tesis con otros autores. Un listado detallado de estas publicaciones se puede ver en la sección A.2 del Apéndice A, y

[4] http://msdn.microsoft.com/en-us/library/cc948344.aspx
[5] http://web.univ-pau.fr/~bruel/Enseignements/IDM/ProjetAspect.html

la versión más actualizada del listado se puede encontrar en el sitio web de la autora (`http://www.lsi.us.es/~reinaqu/colaboraciones.html`). Las colaboraciones que merece la pena destacar son las siguientes:

- [JSS'11] [306] es una revista con un factor de impacto de 1.277 en 2010. En esta publicación se propone CONFIDDENT, un *framework* para el diseño, desarrollo y mantenimiento de listas de control de acceso (ACLs) consistentes y no redundantes para *firewalls*. El *framework* está basado en modelos e inspirado en la arquitectura de `MDA`. La colaboración en este trabajo está relacionada con la infraestructura basada en el `DSDM` para el diseño del *framework*.

- [IJCIS'11] [132] es una revista con un factor de impacto de 1.433 en 2010. En esta publicación se propone Guaraná, un `DSL` para la integración de aplicaciones basado en los patrones de integración definidos por Hohpe [185]. La colaboración en este trabajo ha estado en la definición e implementación del *DSL* usando las herramientas del Eclipse Modelling Project [95] y la implementación de un editor gráfico para definir modelos Guaraná.

- [ECMFA-MELO'11] [143] es una publicación en la que se propone la aplicación de una arquitectura inspirada en `MDA` y una filosofía dirigida por modelos para el desarrollo de bases de datos con restricciones, ya que en este área no hay estándares y es muy dependiente de la tecnología. En este trabajo se ha contribuido en la definición de la arquitectura inspirada en `MDA` y en la parte de metamodelado.

- [OOSPLA-EA'04] [109] y [IADIS-WWW'04] [108] son publicaciones en las que se colaboró aportando el conocimiento y el sesgo del área del `DSOA`.

- [ICWE'03a] [110] y [ICWE'03b] [107] son dos publicaciones interesantes, ya que se publicaron en la conferencia de ingeniería web, aunque en estos casos, la colaboración fue más tangencial.

Citas de otros autores Otros resultados importantes son las referencias que se han realizado a nuestros trabajos, ya que transmiten la relevancia del mismo. En la tabla 1.1 se muestra un resumen de citas por publicación, teniendo en cuenta que no se han contabilizado las autocitas. En el resumen se puede comprobar que la publicación más citada es [332], donde se presenta el germen de `MWACSL`, seguida de cerca por [317], que aunque es meramente un informe técnico, ha servido como introducción a los conceptos que se manejan en orientación a aspectos a la comunidad hispano-hablante.

Un listado detallado de estas citas y clasificado por tipo se muestra en el Apéndice B y la versión más actualizada del mismo se puede encontrar en la página web de la autora (`http://www.lsi.us.es/~reinaqu/citaciones.html`). La tabla 1.2 muestra un resumen detallado de estas publicaciones, clasificando las referencias a nuestros trabajos por tipo de publicación. La fila Otros de la tabla recoge trabajos que no se pueden clasificar como ninguno de los tipos anteriores, como por ejemplo, la referencia a

nuestro trabajo publicada en MSDN Library[6] o la inclusión en el sitio web de Pressman & Associates como bibliografía de referencia para el modelado de aplicaciones web[7] .

Tipo	LSI-TR'00 [317]	GAOP'02 [322]	DDMA'02 [323]	JISBD-DSOA'03 [340]	ECOOP-AAOS'03 [331]	UML-AOM'04 [332]	RCC'04 [324]	ECOOP-MAHCC'05 [326]	JISBD-DSOA'06 [333]	ENTCS'07 [327]	ICWE-AEWSE'07 [335]	Total
Revistas	2	1	3	1	3	3	1	1		1		**16**
Congresos y Talleres	3	2	1		5	17		1		5	1	**35**
Libros	1											**1**
Informes Técnicos	1	2		1	2	2			1			**9**
Tesis	8		1	1	2	3	2	4		1		**22**
Proyectos Fin Carrera	2											**2**
Otros		1						1				**2**
Total	**17**	**6**	**5**	**3**	**12**	**25**	**3**	**7**	**1**	**7**	**1**	**87**

Tabla 1.2: Clasificación por tipo de las publicaciones que referencian a nuestros trabajos

Finalmente, cabe destacar que Robert E. Filman está realizando un trabajo de recolección de artículos y publicaciones relacionados con el ámbito de la orientación a aspectos [118], y en su última versión, publicada en 2005, aparecen seis de nuestros trabajos referenciados [322, 323, 331, 332, 337, 109, 337].

1.5 Organización de la memoria

Esta tesis se ha estructurado en cuatro partes claramente diferenciadas:

Parte I. En esta primera parte se introduce el contexto en el que se desarrolla esta tesis, se presenta y se motiva el problema a resolver. Dentro de este bloque el capítulo 1 introduce el contexto en el que se sitúa esta tesis, define el problema a resolver y presenta un resumen de las contribuciones realizadas, mientras que el capítulo 2 motiva el trabajo presentado en esta tesis y analiza las propuestas existentes indicando cuáles son los problemas de las mismas y cómo mejora nuestra propuesta estos problemas.

Parte II. En esta parte se da una visión general de los conceptos y el estado actual de la tecnología de cada una de las áreas de investigación con las que está relacionada esta tesis. Cada capítulo comienza con una breve introducción y una sección llamada

[6] `http://msdn.microsoft.com/en-us/library/cc948344.aspx`
[7] `http://www.rspa.com/reflib/AnalysisWebApps.html`

Terminología en la que se definen los conceptos básicos y siglas utilizados en el área de investigación objeto del capítulo. Además, la sección *Terminología* se acompaña de un mapa conceptual en el que se muestran las relaciones entre estos conceptos.

En el capítulo 3 se da una visión global del estado del arte en el área del `DSOA`, yendo desde propuestas a nivel de implementación hasta modelado. El capítulo 4 se dedica a introducir la terminología utilizada en el ámbito del `DSDM`. El capítulo 5 describe cómo se modela la navegación en algunas propuestas de ingeniería web. El último capítulo del bloque que comprende el estado del arte presenta como se están aplicando en el campo de la ingeniería web los nuevos paradigmas de desarrollo de software (la orientación a aspectos y el desarrollo dirigido por modelos).

Parte III. En esta parte se dan todos los detalles de la propuesta presentada en esta tesis. El capítulo 7 da una visión general de `MWACSL`, situando la propuesta dentro del marco conceptual definido por Stalh y Völter [363] para el desarrollo de software dirigido por modelos. El capítulo 8 se centra en la navegación a nivel independiente de plataforma. En él presentan dos lenguajes específicos de dominio para separar, por una parte, la navegación estructural y de utilidad, y por otra, los flujos de navegación. El capítulo 9 se centra en la navegación dentro de algunas plataformas específicas. En concreto, se trabaja con `HTML`, `JSP` y Spring Web Flow. El último capítulo del bloque (capítulo 10) presenta la forma de combinar los distintos *concerns* navegacionales separados a nivel independiente de plataforma.

Parte IV. El último bloque, que incluye solamente el capítulo 12, presenta un resumen, concluye la tesis y apunta las líneas de trabajo futuro.

Parte V. En esta parte se añaden una serie de apéndices que dan más detalles acerca de las publicaciones obtenidas como resultado de esta tesis. En el Apéndice A se presenta, por una parte, un listado detallado de las publicaciones obtenidas como resultado directo de esta tesis, y, por otra, otro listado detallado de las publicaciones relacionadas. Finalmente, el Apéndice B presenta un listado detallado de los artículos de otros autores que han citado a nuestras publicaciones.

Motivation is what gets you started.
Habit is what keeps you going.
Jim Ryun

2 Motivación

En este capítulo se presenta la motivación de esta tesis. Para ello, el capítulo se estructura de la siguiente manera: en primer lugar, en la sección 2.1 se introduce el capítulo. Luego, en la sección 2.2 se da una visión general, sin entrar en mucho detalle, de los trabajos relacionados con la propuesta que se presenta en esta tesis. La sección 2.3 compara las distintas propuestas de modelado orientadas a aspectos. Después, el foco se pone en aquellas propuestas de ingeniería web que están aplicando ideas tanto del desarrollo de software dirigido por modelos como del desarrollo de software orientado a aspectos. Estas propuestas se analizan desde tres perspectivas distintas: en la sección 2.4 se estudian las características relacionadas con la orientación a aspectos; en la sección 2.5 se analizan las propuestas desde la perspectiva del desarrollo de software dirigido por modelos; finalmente, se cierra el círculo viendo cómo estas propuestas se enfrentan al diseño de la navegación. El capítulo termina con una sección de sumario.

2.1 Introducción

La Web es un entorno en constante evolución, lo que provoca que en las aplicaciones web tenga lugar frecuentemente un fenómeno conocido como co-evolución. La co-evolución [72, 139] tiene que ver con el hecho de que al introducir un sistema de información en una organización se producen una serie de cambios en sus procesos de negocio, lo que a su vez, también produce cambios en el sistema de información implementado. Los usuarios también encuentran nuevas formas, mejores y más eficientes

de usar el sistema, y solicitan estos cambios. Este conjunto de cambios conducen de nuevo a una evolución cíclica de los procesos de negocio, los usuarios y los sistemas de información [140]. Este fenómeno hace que sea necesario diseñar los sistemas de tal forma que los requisitos puedan evolucionar mientras que el sistema está en uso. Este fenómeno afecta particularmente a las tecnologías de desarrollo de interfaces de usuario y, como consecuencia, a la navegación, un concepto estrechamente relacionado con las interfaces y propio de las aplicaciones web.

En los últimos años, sobre todo desde el área de la ingeniería web, se ha realizado mucho trabajo para separar el modelado de la navegación del modelado conceptual. Para conseguir esta separación, la gran mayoría de las propuestas de ingeniería web definen el modelo de navegación como una vista del modelo conceptual, lo que implica que existe un gran acoplamiento entre ambos modelos, provocando, por una parte, que un cambio en el modelo conceptual interfiera en el modelo de navegación, y, por otra, que no se pueda definir el modelo de navegación hasta que no se haya definido el modelo conceptual por completo. Sin embargo, la navegación es un dominio complejo que puede descomponerse en diferentes *concerns* [197]. Algunos de estos *concerns* navegacionales, como la navegación estructural y de utilidad, o los flujos de navegación controlada, se pueden definir de forma totalmente independiente del modelo conceptual, lo que permite adelantar su diseño, y desacoplarlos del modelo conceptual. Sin embargo, aunque algunas propuestas de ingeniería web han comenzado a aplicar la filosofía de la orientación a aspectos, tal y como se verá más adelante en el capítulo, ninguna de ellas separa de forma explícita la navegación estructural y de utilidad, y solamente algunas lo hacen con los flujos de navegación.

2.2 Visión General de los trabajos relacionados

Para enfrentarse a la problemática presentada en la introducción se puede acudir a trabajos relacionados en varias disciplinas: el modelado orientado a aspectos, el desarrollo de software dirigido por modelos, los lenguajes específicos de dominio y la ingeniería web. Se puede decir que estas disciplinas no son compartimentos estancos, y cada vez surgen más trabajos que se enmarcan en varias de estas áreas. En esta sección se va a dar una visión general de los trabajos que se están haciendo en las mismas, centrándose sobre todo en aquéllos que están situados en varias de estas áreas. En los siguientes apartados se analizarán con más detalle estos trabajos.

La primera gran área a tener en cuenta es el modelado orientado a aspectos. A grosso modo se puede decir que las propuestas de modelado orientadas a aspectos se pueden clasificar en dos grandes grupos: aquéllas cuyo objetivo es modelar programas que ya se han implementado en una plataforma orientada a aspectos, como [173], [303] y [111]; y las que se centran en estructurar los modelos de acuerdo a los diferentes aspectos que componen un sistema (en este grupo se encuentran [60] y [131]). Desde nuestro punto de vista, no hay que descartar ninguna, aunque deben ser empleadas con diferentes

objetivos. Está claro que el uso de modelos permite elevar el nivel de abstracción, con lo cual, podemos razonar mejor sobre los problemas. Teniendo esto en cuenta, creemos que para separar aspectos a nivel de modelos, hay que estructurarlos. Esto no quiere decir que se descarten las otras propuestas, ya que los modelos pueden recoger distinto nivel de detalle. Así que, considerando los lenguajes de programación orientados a aspectos como nuevas plataformas de implementación, algunas de las propuestas del segundo grupo, tales como [173] y [303] se pueden utilizar a nivel de modelos específicos de plataforma con objeto de poder generar código para AspectJ [309] e HyperJ [71], respectivamente.

Las propuestas de modelado orientadas a aspectos también se pueden clasificar en simétricas o asimétricas [175], dependiendo del tipo de elementos a componer, aunque recientemente algunas de las propuestas han evolucionado (como [60]), y proporcionan los dos tipos de composición, en cuyo caso se conocen como híbridas.

En relación al área de los Lenguajes de Aspectos Específicos de Dominio (`LAEDs`) existen muchas propuestas, como las publicadas en los talleres denominados Domain-Specific Aspect Languages Workshops [2, 1]. Muchos de los artículos publicados en estos talleres se centran en la especificación de un nuevo lenguaje de aspectos para un dominio concreto. En general, estos lenguajes se definen en el espacio tecnológico de las gramáticas (o *grammarware*) [228], y no en el espacio tecnológico de los modelos (o *modelware*). Sin embargo, recientemente han aparecido AOM-DSML [188] y `MWACSL`, la propuesta recogida en esta tesis, que trabajan con `DSL's` en el espacio tecnológico de los modelos.

Por otra parte, como nuestra propuesta está orientada al desarrollo de aplicaciones web, poniendo especial interés en el aspecto de la navegación, es necesario hacer referencia a muchas de las propuestas de ingeniería web, ya que actualmente muchas de ellas proponen un diseño de la navegación separado. De manera general puede decirse que la navegación se trata desde dos perspectivas distintas: desde el punto de vista del comportamiento y desde un punto de vista estructural. Por una parte, las propuestas que diseñan la navegación desde la perspectiva del comportamiento se basan en la semántica operacional de la navegación, es decir, exploran el estado en el que quedará el sistema cuando un evento provoca que se ejecute una transición. En otras palabras, indican cuál será el estado del sistema tras un evento. Por otra parte, las propuestas centradas en perspectivas estructurales se centran en estructurar la información en contextos que representan trozos de información coherente interconectados mediante enlaces con significado.

El modelado de la navegación desde la perspectiva del comportamiento se basa en máquinas de estado, mientras que los enfoques estructurales se centran en las relaciones estructurales y de composición entre las estructuras de navegación. Por último, hay propuestas de ingeniería web que están mezclando orientación a aspectos y desarrollo dirigido por modelos. La mayoría de ellas se basan en extender una propuesta no orientada a aspectos para introducir un nuevo tipo de *concern*. Así se han definido extensiones orientadas a aspectos para tratar con el aspecto de adaptación para `UWE` [385, 181, 418],

Hera-S [187, 56] y WebML [58, 347], todas ellas siguiendo una aproximación asimétrica. OOWS [123, 386] y OOHDM [354, 146], sin embargo, están siendo revisados para separar conceptos mediante una aproximación simétrica. Desde otra perspectiva, ya que se mueve en el espació tecnológico de las gramáticas en lugar del de los modelos, WebDSL [392] es una propuesta para modelar aplicaciones web dinámicas que aplica la separación de conceptos usando lenguajes específicos de dominio, siguiendo un enfoque híbrido.

2.3 Análisis de las propuestas de Modelado Orientado a Aspectos

Uno de los foros más específicos de difusión de trabajos relacionados con el modelado orientado a aspectos es el Workshop on Aspect-Oriented Modelling (AOM) que se ha celebrado en dos ediciones, una, junto a la *International Conference on Aspect Oriented Software Development (AOSD)* y otra, junto con la *International Conference on Model-Driven Engineering, Languages, and Systems (MODELS)*. Los artículos, resúmenes y presentaciones de las distintas ediciones del taller están disponibles en [16]. En una de las ediciones de 2008 se echó la vista atrás para revisar las motivaciones, retos y objetivos del modelado orientado a aspectos [213]. Una de las conclusiones fue que durante las ediciones anteriores la investigación en este área se había centrado principalmente en tres campos:

(1) La búsqueda, a nivel de modelado, de diferentes abstracciones que sean adecuadas para modelar los constructores que aparecen en los lenguajes de programación orientados a aspectos como por ejemplo, los constructores punto de corte y aspecto.
(2) El llevar los beneficios de la orientación a aspectos al modelado y al desarrollo dirigido por modelos, y viceversa.
(3) La especificación de *frameworks* de validación y verificación para software orientado a aspectos. Aquí, los problemas estudiados van desde la detección de conflictos entre diferentes aspectos hasta la confirmación de que un aspecto afecta a los puntos de corte adecuados en la aplicación base.

Generalmente estas tres áreas no son excluyentes, y muchos trabajos de investigación abordan varias de ellas, aunque también hay algunos trabajos fuera de estas tres grandes líneas. En la tabla 2.1 se muestra un resumen de los principales objetivos y retos detectados en el marco de estas áreas. La primera columna hace referencia al área concreta. Se han definido abreviaturas, de tal forma que AOM&AOP hace referencia a la primera línea de investigación dentro del Modelado Orientado a Aspectos (MOA) introducido en el párrafo anterior, AOM&MDSD hace referencia a la segunda línea y, finalmente, AOM&VV se refiere a la última.

Campo	Objetivo/Reto
AOM&AOP	**Objetivo:** • Proporcionarle a los desarrolladores de software orientado a aspectos las maneras apropiadas para facilitar el análisis de sus problemas así como el diseño de sus soluciones.

Tabla 2.1: Objetivos y retos de las áreas de investigación dentro del MOA.

Campo	Objetivo/Reto
	Reto: • Encontrar el nivel de abstracción correcto para que los medios de modelado proporcionados sean adecuados tanto para una gran cantidad de problemas como para una gran variedad de lenguajes de programación orientados a aspectos.
AOM&MDSD	**Objetivos:** • Determinar las similitudes, diferencias y relaciones entre la transformación de modelos, la composición de modelos y el *weaving* de modelos. • Determinar hasta qué punto las técnicas de transformación tradicionales pueden usarse para implementar *weavers* de modelos orientados a aspectos. • Liberar a los desarrolladores de software de la necesidad de usar lenguajes de programación orientados a aspectos.
AOM&VV	**Objetivos:** • Detectar conflictos entre aspectos. • Confirmar si los aspectos afectan a los puntos de corte correctos en la aplicación base. • Revelar la presencia de *crosscutting concerns* en el software. **Retos:** • Determinar qué formalismos deben usarse. • Determinar cómo los formalismos pueden adaptarse para la orientación a aspectos. • Determinar cómo se puede facilitar la aplicación de estos formalismos a los desarrolladores ordinarios de software orientado a aspectos.

Tabla 2.1: Objetivos y retos de las áreas de investigación dentro del MOA.

En el resto de la sección se analizan con más detalle las diferentes propuestas para especificar conceptos a nivel de diseño. El análisis es una actualización del trabajo presentado en [332]. Como resultado del mismo se presenta la tabla 2.2, que muestra un resumen de las principales características analizadas. En esta tabla se ordenan las diferentes propuestas por su año de publicación. En la primera columna de la tabla se escribe un acrónimo que hace referencia al nombre de la propuesta; en la segunda columna aparecen los autores de la misma; la tercera columna muestra la referencia más interesante en la que se explica la aproximación; finalmente, la cuarta columna contiene el año de la primera publicación de la propuesta, que contrasta con el año en que se produce la última publicación. Mirando estas dos columnas se puede tener una idea de la vigencia de la propuesta. El resto de columnas reflejan un conjunto de características que se explican con más detalle a continuación:

Nivel de abstracción (NA). Esta propiedad puede tener dos valores: alto o bajo. El nivel de abstracción representa si los constructores propuestos en el lenguaje de modelado son cercanos a los conceptos que se manejan en los lenguajes de programación. Una propuesta con un bajo nivel de abstracción usa constructores muy cercanos a los utilizados en los lenguajes de programación. El nivel de abstracción alto se representa con una flecha hacia arriba (⇑), mientras que el bajo se representa con una flecha hacia abajo (⇓).

Inspirada por (Inspir). Muchos de los lenguajes de modelado se han inspirado en propuestas que separan conceptos en programación. Esta columna refleja el nombre de la propuesta que ha inspirado a los conceptos que se manejan en el lenguaje de modelado. Aquí aparecen valores como Aspect/J, Hyper/J, *Subject-Oriented*

Programming (SOP), Java Aspect Components framework (JAC) y n/a, indicando, en este último caso, que no existe una aproximación concreta en la que se haya inspirado la propuesta.

Mecanismo de extensión (MecExt). Muchas de las propuestas usan UML como lenguaje base, y proponen diferentes extensiones a UML para expresar los aspectos o *concerns*. Esta columna representa la clase de mecanismo de extensión utilizado, bien sea un perfil UML, un metamodelo u otros, como estereotipos (ester.), valores etiquetados (val. eti.), restricciones (restr.) u ocurrencias de patrones de diseño (oc. pat. dis.), en el caso de las propuestas más antiguas. Esta columna también puede contener el valor n/a si la propuesta de modelado no usa UML.

Propósito (PRO). Esta propiedad puede tener dos valores diferentes: general (▽) o específico (▼). Una propuesta de propósito general es aquélla que define un lenguaje de modelado para tratar con cualquier clase de aspecto, mientras que una propuesta de propósito específico necesita definir un lenguaje para cada tipo de aspecto.

Tipo (TIP). Esta propiedad puede tener tres valores: simétrico (representado en la tabla 2.2 por ♦♦), asimétrico (◊♦) o híbrido (◊♦♦).

Acrónimo	Autores	Ref.	Año prim.	Año últ.	NA	Inspir	MecExt	PRO	TIP
Estereotipo	J. Suzuki et al.	[368]	1999	1999	⇓	Aspect/J	ester.	▽	◊♦
	J. L. Herrero et al.	[182]	2000	2002	⇓	n/a	ester.	▼	◊♦
Metamodelo	C. v. F. G. Chavez et al.	[395]	1999	2002	⇓	Aspect/J	MM	▽	◊♦
	Y. Han et al.	[173]	2004	2005	⇓	Aspect/J	MM	▽	◊♦
	I. Philippow et al.	[303]	2003	2003	⇓	Hyper/J	perfil	▽	♦♦
	J. M. Lions et al.	[236]	2002	2002	⇓	Aspect/J	MM	▽	◊♦
AOCE	J. Grundy	[165]	1999	2006	⇑	n/a	ester.	▽	◊♦
UMLAUT	W. M. Ho et al.	[184]	1999	2002	⇑	n/a	ester. val. et. Oc. Pat. Dis.	▽	♦♦
CAM/DAOP	M. Pinto et al.	[304]	2001	2005	⇑	n/a	ester.	▽	◊♦
AOP2UML	M. M. Kande et al.	[208]	2001	2002	⇓	Aspect/J	ester.	▽	◊♦
Profile	A. A. Zakaria et al.	[416]	2002	2002	⇓	Aspect/J	perfil	▽	◊♦
	M. Basch et al.	[28]	2003	2002	⇓	Aspect/J	perfil	▽	◊♦
	J. Evermann	[111]	2007	2007					
ADM	J. P. Barros et al.	[27]	2002	2003	⇑	n/a	profile	▼	◊♦
AODM	D. Stein et al.	[364]	2002	2006	⇓	Aspect/J	ester.	▽	◊♦
Theme/UML	S. Clarke et al.	[60]	2002	2006	⇑	SOP Hyper/J Aspect/J	MM	▽	♦♦
SUP	O. Aldawud et al.	[8]	2002	2005	⇓	Aspect/J	perfil	▽	◊♦
AAM	R. France et al.	[131]	2002	2006	⇓	n/a	MM	▽	◊♦ ♦♦
CoCompose	D. Wagelaar et al.	[402]	2002	2006	⇑	n/a	MM	▽	♦♦
UFA	S. Herrmann	[183]	2002	2002	⇑	Aspectual Collaborat.	MM	▽	◊♦
UML4AO	R. Pawlak et al.	[302]	2002	2005	⇓	JAC	perfil	▽	◊♦

Tabla 2.2: Evaluación de las propuestas de MOA.

Acrónimo	Autores	Ref.	Año prim.	Año últ.	NA	Inspir	MecExt	PRO	TIP
AVA	M. Katara et al.	[210]	2003	2006	⇑	superimpos.	ester.	▽	♦♦
AML	I. Groher et al.	[155]	2003	2004	⇓	Aspect/J	ester. val. eti. restr.	▽	◊♦
Stratified Frameworks	T. Kühne et al.	[21]	2003	2006	⇑	Architecture Stratification	n/a	▽	◊♦♦
AOSDUC	I. Jacobson	[198]	2003	2007	⇑	Hyper/J Aspect/J	ester.	▽	◊♦
MDA	S. J. Mellor V. Kulkarni et al.	[246] [227]	2003 2003	2003 2003	⇑ ⇑	n/a n/a	perfil n/a	▽ ▽	
AODM	J. Gray et al.	[150]	2003	2011	⇑	n/a	n/a	▽	◊♦
Protocol Modeling	A. McNeile et al.	[243]	2003	2010	⇑	mixins	n/a	▽	♦♦
AOMDF	R. France et al.	[131]	2004	2006	⇑	n/a	MM	▽	◊♦♦
VC	A. Muller	[263]	2004	2006	⇑	n/a	profile	▽	◊♦
IDAM	W. Coelho et al.	[63]	2004	2006	⇓	Aspect/J	n/a	▽	◊♦
AOSF	M. Mahoney et al.	[240]	2004	2010	⇑	n/a	n/a	▽	◊♦
JPDD	D. Stein et al.	[365]	2004	2011	⇑	Aspect/J	profile	▽	◊♦
MWACSL	A. M. Reina et al.	[332]	2004	2010	⇑	n/a	n/a	▽	◊♦♦
UML Bottom Up	M. Tkatchenko et al.	[381]	2005	2005	⇑	AspectJ	MM	▽	◊♦
MWEAVR	T. Cottenier et al.	[73]	2005	2007	⇑	n/a	perfil	▽	◊♦
SBWS	J. Klein et al.	[214]	2005	2007	⇓	n/a	n/a	▽	◊♦
MATA	J. Whittle et al.	[406]	2007	2009	⇑	n/a	n/a	▽	◊♦
RAM	J. Kienzle et al.	[212]	2007	2010	⇑	n/a	MM	▽	◊♦♦
HiLa	J. Zhang et al.	[417]	2007	2011	⇑	JasCo	MM	▽	◊♦
AO Executable UML 2.0	L. Fuentes et al.	[134]	2007	2009	⇑	CAM/DAOP	perfil	▽	◊♦
Smart Adapters	B. Morin et al.	[230]	2007	2010	⇑	n/a	MM	▽	◊♦
GReCCo	A. Hovsepyan et al.	[189]	2008	2009	⇑	n/a	n/a	▽	♦♦
KerTheme	O. Barais et al.	[23]	2008	2010	⇑	SOP Hyper/J Aspect/J	MM	▽	◊♦♦
GeKo	B. Morin et al.	[262]	2008	2010	⇑	n/a	MM	▽	◊♦
AOM-DSML	A. Hovsepyan et al.	[190]	2009	2010	⇑	n/a	n/a	▼	♦♦
Aspect Weaver UML	M. Nouh et al.	[276]	2009	2010	⇑	n/a	perfil	▽	◊♦

Tabla 2.2: Evaluación de las propuestas de MOA.

Si analizamos la tabla 2.2, se puede ver que una de las primeras propuestas en incorporar aspectos a la etapa de diseño fue [368], donde se proponía una extensión al metamodelo de UML para dar soporte a los aspectos. Junto con esta propuesta se ha agrupado a [182], ambas bajo el nombre Estereotipo, ya que utilizan este mecanismo para extender UML y poder expresar aspectos. Al evolucionar UML, se definen mecanismos más formales para su extensión. Así, en las propuestas siguientes utilizan extensiones *ligeras*, que se han agrupado bajo el nombre Profile [416, 28, 111], o extensiones *pesadas*, agrupadas bajo el nombre Metamodelo [395, 173, 303, 236]. Por último, se ha definido

también un grupo de propuestas que aplican las ideas presentadas en MDA utilizando las transformaciones de modelo como forma de hacer el tejido (*weaving*) a nivel de modelos, englobando a [246] y a [227].

En [246] se propone un *framework* para incorporar conceptos de la orientación a aspectos en el modelado y en MDA. El tejido de los aspectos se realiza mediante transformaciones de modelos, y, finalmente, se obtiene un modelo que incluye estructura, comportamiento y lógica. Los modelos ejecutables se definen con el perfil Executable UML [247].

Por otra parte, en [227], se propone el uso de MDA y la especificación de componentes y aspectos en su propio lenguaje de modelado. También ven el tejido de un aspecto en el componente como una transformación del modelo del componente. Así, en este contexto un tejedor es un transformador de modelos que toma como entrada el modelo del aspecto y el modelo del componente y produce el modelo del componente transformado. También resalta la necesidad de investigar cómo modelar los aspectos.

En [28] se incorporaron constructores nuevos a UML (puntos de corte y aspectos) y se propone el uso de paquetes para separar y encapsular aspectos. Además, en [155] se propone una extensión a UML mediante la definición de estereotipos, valores etiquetados y restricciones.

Resumiendo la tabla 2.2 se tiene que las mayoría de las propuestas más antiguas trabajan con constructores muy cercanos a los definidos en los lenguajes de programación, siendo éstos inspirados principalmente por Aspect/J. Las propuestas más modernas se van alejando de la implementación, y se centran en estructurar modelos, incorporando muchas de ellas ideas del DSDM. Finalmente, la mayoría de ellas usan un lenguaje de aspectos de propósito general, es decir, mediante un único metamodelo o perfil, se expresa cualquier tipo de aspecto. Solamente hay tres propuestas, AODM, AOM-DSML y MWACSL que trabajan con lenguajes específicos de dominio.

2.4 Análisis de las propuestas de Ingeniería Web en relación al DSOA

Los investigadores que trabajan en el área de la ingeniería web se han percatado de las ventajas que les pueden aportar las ideas y conceptos que se manejan en el área del DSOA. Así, en los últimos años, algunas de las propuestas de este ámbito están aplicando algunas de las ideas promulgadas en el DSOA. En este apartado se define un marco de trabajo que permite analizar cómo estas propuestas aplican las ideas de la orientación a aspectos. La mayoría de ellas extienden a una propuesta de ingeniería web ya existente para que sea capaz de manejar los *crosscutting concern*. En comparación con el número de propuestas de ingeniería web existentes, todavía son pocas las que aplican la orientación a aspectos. De esta forma solamente se han encontrado ocho propuestas que usan ideas orientadas a aspectos, y que aparecen en las columnas de la tabla 2.3, aspectWebML, UWE, SEAL, una extensión de OOHDM, una extensión de OOWS,

WebDSL, FARNav y WAAT. La última columna se reserva a MWACSL, la propuesta presentada en esta tesis. El conjunto de características analizadas, se detallan a continuación:

Propuesta Extendida. Muchas de las propuestas para aplicar las ideas de la orientación a aspectos a la ingeniería web no comienzan de cero, sino que extienden a una propuesta de modelado web ya existente. Algunas de las extensiones han sido diseñadas por los autores originales de las propuestas de ingeniería web, tales como las de OOHDM, UWE y OOWS; otras han sido propuestas por un conjunto de autores diferentes a los originales, como aspectWebML. Finalmente, FARNav, WAAT, WebDSL y MWACSL no se basan en ninguna propuesta anterior.

Año de Primera Publicación. Este criterio proporciona el año en que se introdujeron las ideas de Orientación a Aspectos a las propuestas de desarrollo web. Todas ellas son bastante jóvenes (la más antigua data de 2004).

Año de Última Publicación. Este criterio refleja el año en que se publicaron los últimos artículos en relación con la propuesta y la orientación a aspectos, para tener una idea de la vigencia de la misma.

Enfoque de Separación Avanzada de Conceptos (SoC). En la bibliografía se pueden encontrar dos aproximaciones distintas a la hora de realizar la composición del software [175]: asimétrica y simétrica. Los enfoques asimétricos usan, al menos, dos tipos distintos de elementos a componer, los componentes o clases y los aspectos. Se dice que los componentes o clases representan el núcleo de la funcionalidad o funcionalidad básica, mientras que los aspectos describen comportamientos adicionales. Los enfoques simétricos solamente trabajan con un tipo de elemento a componer, de tal forma que no hay diferencia entre los conceptos que forman el sistema. Mientras que en los enfoque asimétricos los aspectos se tejen en la funcionalidad básica, en los enfoques simétricos los conceptos se componen. También hay propuestas que trabajan con los dos tipos de composición, en cuyo caso se habla de propuestas híbridas. En la tabla una propuesta simétrica se representa por ♦♦, una asimétrica por ◊♦ y una híbrida por ◊♦♦.

***Concerns* Separados.** Este criterio cubre los *concerns* que han sido estudiados en las diferentes propuestas. Así, tanto SEAL como aspectWebML se centran exclusivamente en la personalización. SEAL está enfocado en las propiedades del usuario y del dispositivo, mientras que aspectWebML también tiene en cuenta propiedades de localización, tiempo y de red. En UWE se han estudiado el control de acceso y la navegación adaptable como aspectos. Los autores presentan aspectos para la ocultación de enlaces adaptables y la generación de enlaces adaptables. Rossi et al. [197, 141], por otra parte, han estudiado *concerns* relacionados con diferentes tipos de aplicaciones web, tales como *concerns* estructurales en las aplicaciones de hipermedia física y la conciencia en las aplicaciones de trabajo en grupo. También han estudiado *concerns* más generales, como los volátiles, que suelen aparecer en

aplicaciones web de comercio electrónico, o los navegacionales. OOWS también trata *concerns* navegacionales. En WebDSL se definen sublenguajes para tratar con el modelo de datos, la interfaz de usuario, el control de acceso, los *workflows* y la validación de datos. FARNav se centra específicamente en separar el enrutamiento de la navegación en aplicaciones J2EE. WAAT es una propuesta de reingeniería, y por lo tanto, los conceptos no están separados, sino que son el resultado de un proceso de minería de *concerns*. En este caso los *concerns* son las partes del software responsables de una tarea, concepto u objetivo. Finalmente, aunque MWACSL es una propuesta más general, ha estudiado los flujos de navegación y la navegación estructural y de utilidad.

Extensión General Orientada a Aspectos. Hay algunas extensiones que se han diseñado para manejar un aspecto o *concern* específico y son difíciles de extender para tratar con otros aspectos o conceptos. Así, por ejemplo, FARNav define una notación para separar el enrutamiento de la navegación a nivel de modelado, pero no define una notación de modelado para tratar con otros aspectos. WebDSL y MWACSL también trabajan con lenguajes específicos para los *concerns* que separan.

SoC en .... Representa en qué etapa se ha usado la separación de conceptos. Por ejemplo, UWE solamente trata aspectos en el modelado de la navegación, pero no lo hace ni a nivel conceptual ni a nivel de presentación.

Elementos a Componer. Este criterio indica la clase de *concerns* que se han tenido en cuenta para componer.

Consideración de la Semántica de Composición. Esta característica indica si la propuesta considera la composición de conceptos a nivel de modelado para explotar el soporte de las herramientas existentes para componer modelos. En relación a las propuestas evaluadas, la semántica de composición no se explica ni en UWE, ni en FARNav, ni en WAAT. En SEAL no se considera a nivel de modelado, sino a nivel específico de plataforma y, finalmente, se ha tenido en cuenta en OOWS a nivel de requisitos, y en OOHDM, a nivel de modelos de presentación y navegación. Finalmente, la semántica de composición de aspectWebML se ha detallado en [347].

Interacción de Aspectos. Una de las cuestiones abiertas detectadas en el área del Desarrollo de Software Orientado a Aspectos son los problemas que pueden aparecer cuando dos aspectos o conceptos diferentes interactúan [90]. Este criterio refleja si en la propuesta se han tenido en cuenta posibles conflictos entre aspectos.

Herramienta de soporte. Muchas de las propuestas de ingeniería web proporcionan una herramienta para darle soporte a la misma, pero este criterio especifica si las ideas de Orientación a Aspectos se han cubierto en estas herramientas.

CARACTERÍSTICAS	aspectWebML	UWE	SEAL	OOHDM ext	OOWS ext	WebDSL	FarNav	WAAT	MWACSL
Propuesta Extendida	WebML	UWE	Hera-S	OOHDM	OOWS	-	-	-	-
Año Primera Publicación	2006	2005	2006	2005	2007	2007	2004	2005	2004
Año Última Publicación	2008	2008	2009	2010	2007	2011	2007	2005	2010
Enfoque SoC	◊♦	◊♦	◊♦	♦♦	♦♦	◊♦♦	◊♦	♦♦	◊♦♦
Concerns Separados									
• Personalización	✓		✓						
• Control Acceso		✓				✓			
• Nav. Adaptable		✓							
• Con. Navegacionales			✓	✓					
• Con. Volátiles			✓						
• Con. Estructurales			✓						
• Con. Trabajo en Grupo			✓						
• Validación datos						✓			
• Enrutamiento Navegación							✓		
• Trozos Software								✓	
• Nav. Estructural									✓
• Nav. Utilidad									✓
• Flujos Navegación									✓
Extensión de MOA General	✓			✓		✓		✓	
SoC en Requisitos				✓	✓				
SoC en Modelo Conceptual	✓			✓		✓			
SoC en Modelo Navegación	✓	✓	✓	✓			✓		✓
SoC en Modelo Presentación				✓					
SoC en Implementación				✓			✓	✓	
Elementos a Componer									
• Eltos. estructurales	✓								
• Eltos. comportamiento	✓								
• Clases navegacionales		✓							
• Eltos. Mod. Aplicación			✓						
• Tareas					✓				
• Plantillas Información					✓				
• Sublenguajes						✓			
• Aspectos							✓		
• Trozos Software								✓	
• Metamodelos									✓
Consideración Semántica Composición	✓				✓	✓		✓	✓
Interacción Aspectos						✓			✓
Herramienta de Soporte						✓			

Tabla 2.3: Características relacionadas con la SoC.

Si analizamos el conjunto de características representadas en la tabla 2.3, podemos observar que la aplicación de las ideas de la orientación a aspectos a la ingeniería web es una tendencia bastante reciente, datando la publicación más antigua de 2004. Por lo tanto, las extensiones orientadas a aspectos aún tienen que integrarse completamente en las propuestas de ingeniería web. Esta falta de integración puede observarse sobre todo en el criterio Herramienta de Soporte, ya que solamente una las propuestas integra la extensión orientada a aspectos o le da soporte a la misma.

Otro problema importante que se da en este ámbito es la falta de estudio de las interacciones entre aspectos. Aunque como se mostró en la sección anterior, éste es un problema que también está abierto en la comunidad de modelado orientado a aspectos. Las extensiones analizadas se enfocan solamente a un conjunto concreto de aspectos o conceptos, y no tienen en cuenta cómo interactúan unos con otros. Solamente `WebDSL` y `MWACSL` presentan algún tipo de estudio de interacción.

Finalmente, hay que destacar que no hay una tendencia clara en el enfoque de separación seguido. Hay tres propuestas que siguen un enfoque simétrico, cuatro que siguen una aproximación asimétrica y dos híbridas (aunque aspectWebML se podría generalizar para tratar con una descomposición asimétrica). Finalmente, los enfoques asimétricos evaluados son difíciles de extender para tratar con aspectos distintos a aquéllos para los que fueron pensados.

2.5 Análisis de las propuestas de Ingeniería Web en relación al DSDM

Otra de las tendencias más notables en la ingeniería web es la aplicación del desarrollo dirigido por modelos. Como consecuencia, muchas de las propuestas evaluadas en el apartado anterior han ido adoptando esta filosofía de desarrollo. Las características analizadas en este apartado están relacionadas con el `DSDM`, y se resumen en la tabla 2.4. El conjunto de características tiene un doble propósito: (1) comprobar cómo en las propuestas analizadas se están aplicando las ideas del `DSDM`, y (2) ver si las extensiones orientadas a aspectos están integradas dentro del desarrollo dirigido por modelos.

Para conseguir estos objetivos se evalúa el siguiente conjunto de características:

Especificación de Metamodelos. Uno de los elementos claves en el desarrollo de software dirigido por modelos son los metamodelos. Así, es necesario definir metamodelos que especifiquen los constructores utilizados en las notaciones propias y sus relaciones. En este criterio se comprueba si se han definido los metamodelos de apoyo necesarios para la propuesta. Algunos de ellos han sido definidos por autores distintos de los de la propuesta original. Por ejemplo, en `WebML` no se define explícitamente un metamodelo, pero los autores de aspectWebML han definido uno que es la base de la extensión a `WebML` que ellos proponen. En el caso de `WebDSL`, que es una propuesta que trabaja en el espacio tecnológico de las gramáticas, en lugar de utilizar metamodelos para definir la sintaxis abstracta de los lenguajes, el formalismo utilizado es la gramática.

Metamodelo de la Extensión Orientada a Aspectos. Esta característica indica si la extensión para tratar con las ideas de la orientación a aspectos se ha metamodelado.

Lenguaje de Especificación de la Sintaxis Abstracta. En general, hay dos aproximaciones diferentes para especificar metamodelos, definir el metamodelo desde cero, o hacerlo como un perfil UML. En este criterio se indica el mecanismo elegido para especificar el metamodelo. Casi todas las propuestas evaluadas usan MOF para especificar los metamodelos. Solamente SEAL y WebDSL, que están basados en Hera-S y gramáticas, respectivamente, se especifican mediante RDF y SDF (un formalismo para definir sintaxis de lenguajes). Ni en OOHDM ni en WAAT se especifica ninguna técnica, puesto que para estas propuestas no se han definido los correspondientes metamodelos. MWACSL usa EMF, la implementación de MOF del proyecto de modelado de Eclipse para especificar los metamodelos.

Separación de Conceptos Integradas en las Transformaciones. Este criterio indica si la separación avanzada de conceptos está completamente integrada en la propuesta. El símbolo $\simeq$ en una celda indica que la separación avanzada de conceptos está parcialmente integrada.

Tipos de Transformaciones. En esta posición de la tabla se pueden encontrar los valores: PIM2PIM, PIM2PSM, PIM2Cod y PSM2Cod, indicando el nivel de transformación soportado por la propuesta. La mayoría de las propuestas evaluadas solamente dan soporte a transformaciones de Modelos Independientes de Plataforma a Código. Solamente, UWE y MWACSL dan soporte a transformaciones de Modelos Independientes de Plataforma a Modelos Dependientes de Plataforma.

CARACTERÍSTICAS	aspectWebML	UWE	SEAL	OOHDM ext	OOWS ext	WebDSL	FarNav	WAAT	MWACSL
Especificación del Metamodelo	✓	✓	✓		✓	✓	✓		✓
Metamodelo de la Extensión OA	✓	✓			✓				
Lenguaje de Especificación de la Sintaxis Abstracta									
• MOF	✓	✓			✓		✓		
• RDF			✓						
• SDF						✓			
• EMF									✓
Separación Avanzada Conceptos Integrada	✓			$\simeq$	$\simeq$	✓			✓
Tipos de transformaciones									
• PIM2PIM		✓			✓				
• PIM2PSM		✓							✓
• PIM2Code	✓		✓	✓	✓				✓
• PSM2Code		✓							✓

Tabla 2.4: Características relacionadas con DSDM.

Si analizamos las características evaluadas en la tabla 2.4, y las relacionamos con las de la tabla 2.3 se puede observar que la mayoría de ellas no mantiene la separación de conceptos en todas las fases del ciclo de vida. Así, por ejemplo, OOWS separa los

conceptos a nivel de requisitos, pero luego, éstos se componen para obtener modelos conceptuales donde todos los conceptos vuelven a estar mezclados. Por otra parte, OOHDM separa los conceptos desde requisitos hasta diseño, pero lo que se genera en la parte de implementación ya tiene los conceptos mezclados.

Otra característica a destacar es que solamente en UWE y MWACSL se definen algunos modelos específicos de plataforma. Todas las propuestas obtienen código directamente de los modelos independientes de plataforma.

2.6 Análisis de las propuestas de Ingeniería Web en relación a la navegación

El objetivo de esta sección es ver cómo las propuestas evaluadas en las dos secciones anteriores se enfrentan al modelado de la navegación, centrándose, sobre todo, en ver cómo se separa de otros conceptos tales como la interfaz de usuario o los procesos de negocio. El conjunto de características seleccionadas se resume en la tabla 2.5, y se detalla a continuación:

Conceptual. Muchas propuestas de ingeniería web separan el modelado conceptual del modelado navegacional. Exceptuando a FARNav y a WAAT, que solamente se centran en la navegación, el resto de propuestas se enfrentan al modelado conceptual. La mayoría usan un modelo orientado a objetos. aspectWebML, debido a que está basado en WebML, describe el modelo conceptual utilizando diagramas Entidad-Relación. OOHDM utiliza diagramas de clase, pero no están basados en UML, ya que permite el uso de atributos con múltiples valores.

Navegación. En este criterio se recoge la clase de notación utilizada para modelar la navegación. Todas las propuesta evaluadas que provienen del área del diseño web usan modelos estáticos, y muchas de ellas tienen notaciones propias. FARNav usa solamente modelos de comportamiento (diagramas de estados junto con una notación propia para representar modelos de enrutamiento de la navegación). Finalmente, UWE y WAAT usan modelos de comportamiento estáticos y dinámicos.

Navegación estructural y de utilidad. Este criterio indica si la navegación estructural y de utilidad se modelan de forma explícita en la propuesta.

Flujos de navegación. Este criterio indica si los flujos de navegación se modelan de forma explícita, haciendo distinción entre los controlados y la navegación libre.

Presentación. Es difícil separar la navegación de la presentación. WAAT y FARNav no tratan la interfaz en absoluto, por lo tanto, no proporcionan ningún modelo. aspectWebML especifica algunas características de la interfaz de usuario a nivel de hipertexto y no proporciona ningún modelo separado para la presentación.

Procesos de negocio. Recientemente, muchas propuestas han incorporado el modelado de procesos de negocio como una forma de tratar con las aplicaciones con estado o también llamadas de navegación controlada. Este criterio especifica la técnica de modelado utilizada en las propuestas evaluadas para enfrentarse a este concepto. Como puede verse en la tabla 2.5, solamente aspectWebML, `UWE` y `OOWS` tratan este concepto. aspectWebML utiliza una notación propia en el modelo de hipertexto que se basa en la semántica del flujo de control. `UWE` modela la navegación controlada mediante diagramas de actividad estereotipados, mientras que `OOWS` utiliza la notación `BPMN`.

CARACTERÍSTICAS	aspectWebML	UWE	SEAL	OOHDM ext	OOWS ext	WebDSL	FarNav	WAAT	MWACSL
Conceptual									
• Diagrama ER	✓								
• Diagrama Clases UML		✓							
• RDF			✓						
• Diagrama Clases OO				✓					
• Diagrama Clases					✓				
• Diagrama Estados					✓				
• Diagrama Secuencias					✓				
• DSL Propietario						✓			
Navegación									
• Notación Propia	✓		✓	✓	✓	✓	✓		✓
• Diag. Clases Estereotipadas		✓						✓	
• Diag. Estados		✓						✓	
• StateCharts							✓		✓
Navegación Estructural y de Utilidad									✓
Flujos de Navegación	✓	✓			✓	✓			✓
Presentación									
• Notación Propia			✓	✓	✓	✓			✓
• Diag. Clases Estereotipadas		✓							
• Diag. Secuencias		✓							
Proceso Negocio									
• Notación Propia (Workflow)	✓					✓			
• Diag. Actividad Estereotipado		✓							
• Diag. Actividad Estereotipado					✓				

Tabla 2.5: Características relacionadas con la navegación.

Analizando las características de la tabla 2.5, se ve una clara tendencia relacionada con la incorporación del modelado de negocio para tratar con la navegación controlada. Igualmente, algunas de las propuestas, como `UWE`, utilizan modelos tanto estáticos como de comportamiento. Sin embargo, la mayoría solamente utilizan modelos estáticos para describir la navegación.

2.7 Sumario

En este capítulo se ha mostrado cómo uno de los problemas más importantes a los que se tiene que enfrentar la industria de desarrollo de software es el cambio y cómo adaptarse a él. Los cambios se producen principalmente desde dos frentes, por una parte, la variabilidad de requisitos de los clientes y, por otra, la constante renovación de los entornos tecnológicos. Para enfrentarse a estos dos frentes de forma conjunta se ha mostrado válida la combinación de dos nuevas filosofías a la hora de enfrentarse al desarrollo del software: la orientación a aspectos y el uso de modelos como elementos clave para guiar el proceso de desarrollo. Además, se ha visto que un ámbito idóneo para probar la combinación de estos paradigmas es el desarrollo web, ya que la problemática se acentúa debido a que tanto los cambios de requisitos como los de tecnología se producen más rápidamente.

En este contexto, se han analizado cuáles son las áreas de investigación que intentan resolver problemas similares: el desarrollo de software orientado a aspectos, el desarrollo de software dirigido por modelos (sobre todo centrándose en la concepción propuesta en MDA), la ingeniería web y los lenguajes específicos de dominio.

El análisis de las propuestas de modelado orientado a aspectos ya se comenzó en [332]. Allí se determinó que la mayoría de las propuestas existentes por entonces eran de bajo nivel, es decir, que se centraban en definir constructores muy cercanos a los de los lenguajes de programación orientados a aspectos. Entonces, se detectó la necesidad de elevar el nivel de abstracción. Tras hacer una revisión de las propuestas actuales, se tiene que cada vez hay más propuestas que elevan el nivel de abstracción. Sin embargo, la mayoría de ellas usa lenguajes de modelado de aspectos de propósito general, definidos como una extensión de UML.

Si nos centramos en el ámbito de la ingeniería web, como un caso particular del modelado orientado a aspectos, se tiene que la aplicación de las ideas de orientación a aspectos al desarrollo web es un hecho relativamente reciente. Solamente se han encontrado ocho propuestas que las aplican. Algunas de ellas no son extensibles, en el sentido que definen una extensión para tratar un aspecto específico y es difícil de modificar para tratar con otros aspectos distintos de aquél para el que se pensó originalmente la extensión. Además, en la mayoría no se tiene en cuenta la posible interacción de aspectos. Finalmente, estas propuestas, siete de la cuales también fueron estudiadas en [319], se han analizado desde el punto de vista del DSDM y de cómo se enfrentan al modelado de la navegación.

Parte II

Antecedentes

Une place pour chaque chose et chaque chose à sa place.

Proverbio Francés

3

Desarrollo de Software Orientado a Aspectos

Este capítulo da una visión general del Desarrollo de Software Orientado a Aspectos, como una nueva filosofía para descomponer los sistemas software. El capítulo se ha estructurado de la siguiente forma: en la sección 3.1 se introduce el capítulo. Luego, en la sección 3.2, se define la nueva terminología que ha surgido en el área de la separación avanzada de conceptos, para que el lector pueda comprender mejor el resto del capítulo. Esta sección concluye con un mapa conceptual donde se muestran estos conceptos y la relación existente entre los mismos. El resto de los apartados se dedican a profundizar un poco en las técnicas de separación de conceptos existentes en las distintas fases del ciclo de vida de un proyecto. Así, la sección 3.3 se centra en la separación de conceptos en implementación; la sección 3.4 explica las técnicas a nivel de modelado, mientras que en la sección 3.5, se da una visión somera de los aspectos en las fases tempranas. Finalmente, la sección 3.6 resume el capítulo.

3.1 Introducción

La evolución en la ingeniería del software ha tenido lugar gracias a la aplicación de la máxima "divide y vencerás", de tal forma que para abordar la complejidad de un sistema se aplica el principio de separación de conceptos [297, 86]. En su forma más general [294], la separación de conceptos se refiere a la habilidad de identificar, encapsular, y manipular solamente aquellas partes del software que son relevantes para un concepto, objetivo o propósito particular. Con la separación de conceptos se disminuye la complejidad al hacer que sea más fácil razonar de forma aislada sobre cada uno de los

conceptos que componen un sistema, y preocupándose, en una etapa posterior, de componer cada uno de esos conceptos o competencias que se han estudiado por separado. Por contra, la operación de composición incrementa su complejidad.

Conforme la ingeniería del software ha ido evolucionando, este principio se ha aplicado de diferentes maneras. Tras una primera etapa en la que el código de los programas era "código espagueti", donde todo se entremezclaba, apareció lo que posteriormente se conoció como *programación funcional*, en la que los sistemas se descomponen en las distintas funcionalidades que los conforman. Con este tipo de descomposición, los datos quedan esparcidos por las distintas funciones en las que se divide el sistema.

Con la aparición de la *programación orientada a objetos* la descomposición de los sistemas se hace en base a los datos, encapsulándose éstos y la propia funcionalidad en las clases. Sin embargo, con este tipo de descomposición es frecuente que conceptos tales como la coordinación, la seguridad, la distribución o la navegación, no estén perfectamente localizados, sino que se entremezclen con el resto de los conceptos del sistema. A este tipo de conceptos que se entremezclan con el resto de aspectos o con la funcionalidad básica se les conoce como *crosscutting concerns*.

La *Programación Orientada a Aspectos (POA)* [211, 268] ha surgido así para mejorar la modularización de los sistemas, como una nueva forma de descomponerlos para que sean más fáciles de desarrollar, adaptar y mantener [64]. Esto se consigue gracias a que los conceptos del sistema están mejor localizados y a que se proporcionan los mecanismos de composición adecuados para que se pueda razonar sobre las competencias de forma separada. Inicialmente, esta nueva forma de modularizar los sistemas surgió a nivel de implementación. Aunque esta filosofía se ha extendido todas las etapas del desarrollo software tomando el nombre genérico de *DSOA* [119]. Así, actualmente hay técnicas para dar soporte a la composición, modularización y abstracción de los conceptos en todo el ciclo de vida de un proyecto software, cubriendo tanto las fase de requisitos y diseño arquitectónico como de diseño detallado, pruebas e incluso mantenimiento y/o evolución.

3.2 Terminología

A la hora de separar y componer conceptos, existen dos aproximaciones distintas dentro del ámbito del DSOA [175] (la figura 3.1 muestra un mapa conceptual con los principales términos manejados por estas dos aproximaciones, así como los usados a lo largo del capítulo). En primer lugar, el enfoque *asimétrico*, utiliza al menos dos tipos de elementos a componer, los componentes o clases y los aspectos. Se dice que los componentes o clases representan la funcionalidad básica de un programa, y los aspectos describen un comportamiento adicional del sistema que tendrá efecto en la funcionalidad básica. Así, por ejemplo, en [211] los componentes son elementos que se pueden describir y ejecutar de forma independiente, mientras que los aspectos son elementos que se describen en relación a otros componentes o aspectos. Por lo tanto, no se pueden ejecutar de forma

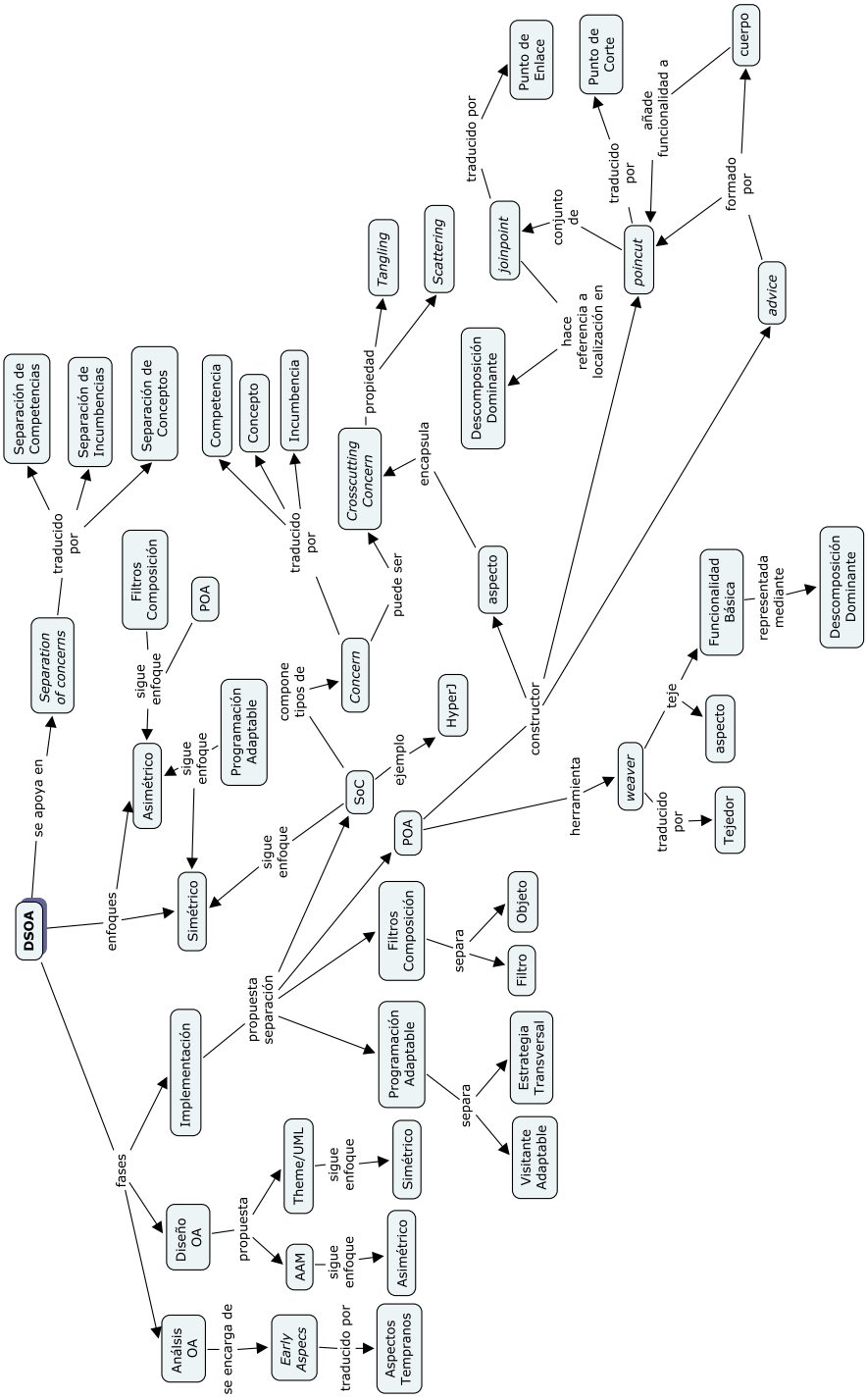


Figura 3.1: Mapa conceptual con la terminología usada en el área del DSOA

independiente.

Por otra parte, el enfoque *simétrico* solamente utiliza un tipo de elemento a componer, de tal forma que la funcionalidad básica también está compuesta por distintos conceptos, no existiendo diferencias entre unos y otros.

En la aproximación asimétrica se dice que los componentes implementan la funcionalidad básica y los aspectos se tejen a la "base" añadiendo funcionalidad, mientras que en la aproximación simétrica todos los conceptos tienen la misma importancia y se utiliza el término composición para combinar todos los conceptos.

De una forma genérica, podemos decir que en el *DSOA* o en la *Separación Avanzada de Conceptos (SAC)* los elementos fundamentales son:

1. Los *elementos que se van a componer*, ya sean aspectos, conceptos, competencias o componentes.

2. Los *puntos de enlace o de unión* (o *join points*) que son aquellos puntos dentro de los elementos que se van a componer en los que se puede realizar la composición. Ejemplos de estos puntos de unión pueden ser los métodos de una interfaz o los miembros de una clase.

3. Las *relaciones de composición* que especifican cómo se va a realizar la composición de los elementos en los puntos de unión.

Además de esta terminología genérica, existen otros términos que dependen de si el enfoque es asimétrico o simétrico. Los términos utilizados en los enfoques asimétricos se representan en la tabla 3.1, mientras que en la tabla 3.2 se resume la terminología utilizada en los enfoques simétricos.

Término	Descripción
Crosscutting	Es un comportamiento que se ejecuta en múltiples situaciones y que se esparce por el código base entremezclándose con otros conceptos.
Advice	Es el comportamiento a ejecutar.
Aspecto	Encapsula el comportamiento a ejecutar (*advice*) y la especificación de dónde se ha de ejecutar ese comportamiento.
Núcleo o base	Parte que contiene la funcionalidad básica y a la que se van a aplicar los aspectos.
Punto de enlace (*joinpoint*)	Un punto concreto en el que se puede ejecutar un comportamiento.
Punto de corte (*pointcut*)	Un predicado que determina un conjunto de puntos de enlace.
Weaving	Es el proceso de aplicar un comportamiento (*advice*) en determinados puntos (puntos de enlace) a la funcionalidad básica.

Tabla 3.1: Terminología utilizada en las propuestas asimétricas

Finalmente, hay que destacar que la comunidad investigadora está haciendo grandes esfuerzos por definir formalmente esta terminología con trabajos como [67, 241, 252, 239, 313]. Además, en el *wiki* [18] de la página web de la comunidad, se puede acceder a un glosario en el que los investigadores definen los principales términos que se manejan en el área.

Término	Descripción
Incumbencia (o *concern*)	Es algún tipo de funcionalidad del sistema. Puede ser una característica o un tipo de procesamiento.
Crosscutting	Es un concepto o incumbencia que se puede ejecutar en múltiples situaciones y cuyo comportamiento se esparce por el código base, entremezclándose con otros conceptos.
Composición	Es el proceso de combinar los conceptos que se han definido de forma separada para conformar el sistema funcionando.

Tabla 3.2: Terminología utilizada en las propuestas simétricas

3.3 Desarrollo de Software Orientado a Aspectos en implementación

La separación avanzada de conceptos se puede enfocar desde diversas perspectivas. La mayoría de las propuestas existentes presentan un enfoque asimétrico. Las principales propuestas para separar conceptos a nivel de implementación son: los Filtros de Composición (`FC`), la Programación Adaptable (`PA`), la Programación Orientada a Aspectos (`POA`) y la Separación Multidimensional de Competencias (`SMC`). En la tabla 3.3 se muestra un resumen de las principales características de estas propuestas. En las columnas se representan las propuestas, mientras que en las filas se representan las características. Así, por ejemplo, los `FC` es una propuesta que sigue una aproximación asimétrica, es decir, que tiene más de un tipo de elemento a componer. Estos elementos a componer son los filtros y los objetos, mientras que los puntos de unión y las reglas de composición se especifican en los propios filtros.

En el resto de subsecciones del apartado se presentan con un poco de más detalle cada una de las propuestas.

Propuesta	FC	PA	POA	SMC
Tipo Aproximación	Asimétrica	Asimétrica->Simétrica	Asimétrica	Simétrica
Elementos a componer	Filtros Objetos	Objetos	Aspectos Clases	Hiperslices
Puntos de unión	Filtros	Estrategia Transversal	Pointcut	Unidades
Reglas de composición	Filtros	Visitante Adaptable	Advice	Estrategias de composición

Tabla 3.3: Propuestas para separar competencias a nivel de implementación

3.3.1 Técnicas basadas en filtros

Los *Filtros de Composición* [5, 36] (o en inglés *Composition Filters*) surgieron intentando expresar la coordinación en los lenguajes orientados a objetos tradicionales. La propuesta consiste en aumentar el modelo orientado a objetos tradicional con *filtros de mensaje*, de tal forma que a los objetos se les asocia una secuencia de filtros. Se puede decir, por tanto, que estos filtros envuelven a los objetos tradicionales. Así, un objeto

de este modelo está compuesto por una capa de interfaz (el envoltorio), y el núcleo del objeto. La capa de interfaz tiene filtros de entrada y filtros de salida, que pueden modificar un mensaje que envíe el objeto o un mensaje que reciba. Los filtros también pueden implementar nuevos métodos que no estuvieran soportados directamente por los objetos originales. En la figura 3.2 se puede ver un objeto envuelto por una interfaz con filtros de entrada y de salida.

Esta propuesta sigue una aproximación asimétrica, ya que los dos elementos a componer (objetos y filtros) se consideran de forma distinta, no teniendo sentido el utilizar los filtros de forma independiente a los objetos. Las relaciones de composición vienen dadas por las especificaciones que se hacen de forma separada donde se enlazan los filtros a los objetos concretos. El mecanismo de composición que se utiliza en este modelo son los filtros del mensaje. En el modelo hay filtros predefinidos, y también se pueden añadir filtros nuevos. De esta manera se pueden capturar conceptos tales como la sincronización o la gestión de errores. Finalmente, los puntos de enlace son los propios filtros, en los que mediante una serie de condiciones y expresiones descritas mediante un lenguaje de patrones, se determina en qué puntos se aplica el filtro.

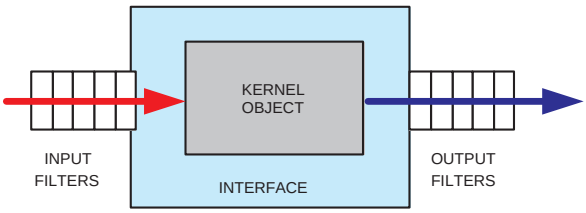


Figura 3.2: Objeto envuelto por una interfaz con filtros de entrada y de salida

3.3.2 Técnicas adaptables

La *Programación Adaptable* [233, 232] (o en inglés *Adaptive Programming*) trata de crear aplicaciones que sean más fáciles de mantener y que puedan evolucionar mejor. La idea subyacente es poder escribir algoritmos que trabajen sobre un conjunto de objetos de una forma tan independiente como sea posible del modelo de clases usado para generar esos objetos. Así se tiene un software que es capaz de adaptarse mucho mejor a los cambios. La capacidad de adaptación se obtiene gracias a programas débilmente acoplados, y para conseguir la pérdida de acoplamiento se separa comportamiento de estructura, de forma que el software adaptable puede trabajar teniendo solamente un conocimiento parcial de la estructura de clases del sistema.

Las clases de un sistema se pueden modelar como un grafo en el que los nodos se corresponden con las propias clases y los arcos son las relaciones entre las mismas. Así, citando a [296]: "un programa adaptable puede verse como un programa orientado a objetos en el que el grafo de clases es un parámetro, y por lo tanto, el grafo de clases

puede cambiarse sin cambiar el programa". De esta forma, un *programa adaptable* está formado por una colección de *patrones de propagación*, que se conocen como *métodos adaptables*. Cada patrón de propagación está compuesto por una serie de *estrategias transversales* y de envoltorios de código. Las *estrategias transversales* seleccionan los objetos que serán afectados por el comportamiento. Se puede decir, por tanto, que en esta propuesta los puntos de enlace son instancias de clases. Los objetos se seleccionan atravesando el grafo de clases, mientras que los envoltorios asocian las acciones a los objetos seleccionados. Un envoltorio se implementa utilizando el patrón visitante, por lo que se le llama *visitante adaptable*.

Resumiendo, podemos decir que una *estrategia transversal* [234] es una descripción de alto nivel de cómo alcanzar a los objetos que participan en un cálculo o una operación, mientras que el *visitante adaptable* indica qué hacer cuando se ha alcanzado a cada uno de los participantes, por lo tanto, se puede considerar que aquí es donde se define el mecanismo de composición. Para alcanzar a un objeto se dará un camino de navegación a través de la estructura de clases. La estrategia transversal es una especificación parcial de uno de estos caminos mencionados antes, de forma que solamente se dan el nodo inicial y el nodo final del camino, pero no se especifica de ninguna forma cómo llegar desde el nodo inicial al nodo final. La principal ventaja de esta especificación parcial es que una modificación de la estructura de clases (es decir, el camino para llegar desde el nodo inicial al nodo final) no tiene ninguna influencia en el resultado de los cálculos.

La figura 3.3 muestra un método adaptable que calcula la suma de los salarios de todos los empleados de una compañía, y cómo el mismo código se puede reutilizar sin ningún cambio para hacer este cálculo con tres grafos de clases distintos (figura 3.3(b)). El código representado en la figura 3.3(a) está tomado de [36], y en él el grafo de clases se define como un atributo estático de la clase `Company`. La estrategia transversal indica los nodos inicial (`Company`) y final (`Salary`) del camino a recorrer, mientras que en el visitante adaptable se indican las operaciones a realizar cuando se alcanzan los objetos. Este mismo programa sería válido tanto para el grafo de clases sencillo representado en la parte superior de la figura 3.3(b), que solamente tiene las clases compañía (`Company`), empleado (`Employee`) y salario (`Salary`), como para otras estructuras más complejas (figura 3.3(b) grafo de clases central o inferior).

Finalmente, cabe destacar que esta propuesta ha ido evolucionando desde una aproximación asimétrica a una aproximación simétrica [175].

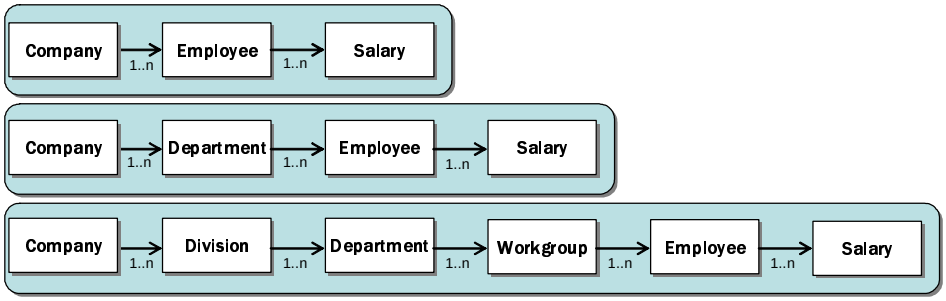
3.3.3 Mecanismos lingüísticos

Otra forma de separar los conceptos que se esparcen o entremezclan por el resto de conceptos del sistema es a través de la creación de nuevos lenguajes o formalismos, en los que se definen nuevas características que permitan representar estos conceptos. Algunos ejemplos de los mecanismos lingüísticos disponibles para representar conceptos son los módulos (clases o paquetes), las anotaciones, la reflexión, o los aspectos.

El *aspecto* es el mecanismo lingüístico que introduce la *POA* [211] para representar

```
import edu.neu.ccs.demeter.dj.ClassGraph;
import edu.neu.ccs.demeter.dj.Visitor;
class Company {
  static ClassGraph cg = new ClassGraph();  // class structure
  Double sumSalaries() {
    String s = "from Company to Salary";    // traversal strategy
    Visitor v = new Visitor() {             // adaptive visitor
      private double sum;
      public void start() { sum = 0.0 };
      public void before(Salary host) { sum += host.getValue(); }
      public Object getReturnValue() { return new Double(sum); }
    };
    return (Double) cg.traverse(this, s, v);
  }
  // ... rest of Company definition ...
}
```

(a) Un método adaptable simple.



(b) Diagramas de clases

Figura 3.3: Método adaptable y diversos grafos de clases para los que es válido

y separar los conceptos que afectan a otros conceptos del sistema [117], es decir, para representar los *crosscutting concerns*. Un concepto se dice que afecta a otros si no se puede expresar de forma monolítica con los mecanismos lingüísticos y la estructuración que proporciona el lenguaje de programación elegido. En [116] se distinguen entre cuatro tipos de conceptos: locales, sistemáticos, combinatorios, y estéticos. La programación orientada a aspectos es útil para manejar los conceptos sistemáticos, que son aquellos que se dispersan por muchos puntos del programa y se encuentra entre las propuestas asimétricas, ya que cuenta con dos tipos de elementos a componer, los aspectos y las clases. Además, estos dos elementos tienen distinta importancia, un aspecto, por sí solo no tiene sentido. Los puntos de enlace vienen determinados por el lenguaje de definición de puntos de corte, y finalmente, el modo en el que el código del aspecto se compone con el de las clases, es decir, la reglas de composición, vienen especificadas en el propio *advice*.

En la mayoría de los lenguajes orientados a aspectos, el aspecto se escribe en el mismo lenguaje que el programa base, mediante un Lenguaje de Aspectos de Propósito General (`LAPG`)(o, en inglés, General-Purpose Aspect Language (`GPAL`)). Un `LAPG` es un lenguaje diseñado para encapsular cualquier tipo de aspecto. Aunque también existe

la opción de que los aspectos se expresen en un lenguaje que use abstracciones más cercanas al dominio del propio aspecto, es decir, usando un Lenguaje de Aspectos Específico de Dominio (LAED). Un LAED (o, en inglés, Domain-Specific Aspect Language (DSAL)) es un lenguaje específico de dominio utilizado para expresar un (*concern*) que se esparce por otras competencias o *concerns*.

A pesar de que los trabajos seminales en el área de la orientación a aspectos estaban relacionados con LAEDs [238], la mayoría de los esfuerzos de la comunidad se han puesto en la investigación en Lenguajes de Aspectos de Propósito General. Sin embargo, de nuevo, los LAEDs están volviendo a ser tema de interés para la comunidad investigadora [61].

3.3.3.1 Lenguajes de Aspectos de Propósito General (LAPG)

AspectJ [309] es uno de los lenguajes de aspectos de propósito general más extendidos. Se ha definido como una extensión a Java que proporciona una serie de mecanismos lingüísticos para capturar y representar los *crosscutting concerns*. Este lenguaje ha hecho populares algunos conceptos, que luego se han adoptado como terminología común dentro de la comunidad investigadora que trabaja con la separación avanzada de conceptos. Así, se denominan *aspectos* (en inglés, *aspect*) a aquellos conceptos que se esparcen por toda la aplicación; *punto de enlace* (o *joinpoint*) a un punto del código del programa donde el comportamiento puede aumentarse con el comportamiento del aspecto, ya sea antes o después del punto de enlace. Un *punto de corte* o *pointcut* define un conjunto de puntos de enlace. Un *advice* es un módulo que contiene la funcionalidad a ejecutar en los puntos de enlace. Finalmente, una *introducción* (o *introduction*), también conocida como declaración inter-tipo es la forma de modificar la estructura estática de la clase afectada por el aspecto, añadiéndole nuevos miembros, cambiando su relación de herencia, o haciendo que implemente nuevas interfaces.

Además de estos conceptos inherentes al lenguaje, AspectJ también ha introducido el concepto de *weaving* o entretejido, como una nueva fase dentro del proceso de desarrollo. Así, la fase de tejido o entrelazado es el momento donde se añade la funcionalidad de los aspectos a la de las clases que componen el sistema. A la herramienta que se encarga de realizar este entrelazado se la conoce como tejedor o *weaver*.

La figura 3.4 muestra un trozo de código que implementa un aspecto AspectJ simple llamado `TestAspect`. El aspecto tiene dos definiciones de puntos de corte, una llamada `outputLog`, y otra llamada `fileLog`. Los puntos de corte expresan un conjunto de puntos de enlace en el código del programa. Para expresar ese conjunto de puntos de enlace se emplea un lenguaje de patrones, así el punto de corte `outputLog` define como puntos de enlace todos aquellos métodos que sean públicos, que devuelvan algo de tipo `void`, cuyo nombre sea `helloWorld` seguido de cualquier secuencia de caracteres, y con cualquier número y tipo de argumentos. Para todos estos puntos de enlace, se aumentará su comportamiento, ejecutando el código definido en el *advice* recuadrado. En este caso, el *advice* es de tipo `before`, lo que indica que el código del bloque asociado al *advice*

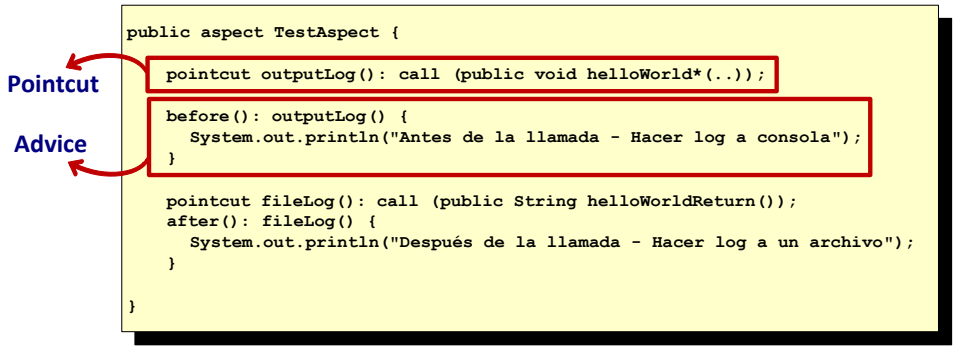


Figura 3.4: Aspecto definido en AspectJ

(la impresión de un mensaje por la consola) se ejecutará justo antes de llamar a los métodos que concuerden con la definición del punto de corte.

3.3.3.2 Lenguajes de Aspectos Específicos de Dominio (LAED)

Normalmente, los LAEDs tienen un nivel de abstracción mayor que los Lenguajes de Aspectos de Propósito General. Además, se puede decir que una propiedad de los LAEDs es que al utilizarlos en una aplicación, se cambia de forma no invasiva, bien el comportamiento, bien la estructura de la misma. La mayoría de los trabajos relacionados con los LAEDs se centran en definir nuevos lenguajes de aspectos para dominios concretos aunque también hay otro grupo de propuestas centradas en plataformas para trabajar con los LAEDs.

En el primer grupo están los trabajos que dieron origen al DSOA, que estaban relacionados con la definición una serie de lenguajes de aspectos específicos de dominio, tales como COOL [237], para la gestión de concurrencia, o RIDL [237] para la serialización de objetos. Otros LAEDs definidos por la comunidad son: RG [249], para el procesamiento de imágenes; AML [193], una extensión a MatLab para el procesamiento de matrices dispersas; AGOL [13], para tratar con sistemas multiagentes; KALA [112], para la gestión de transacciones; ERTSAL [360] para sistemas en tiempo real; ALPH [266] para sistemas relacionados con el dominio de la salud; y DDL [359], para tratar con la distribución.

En el grupo de trabajos relacionados con las plataformas para poder trabajar con los LAEDs están: XAspects [358], un mecanismo de *plugin* para LAEDs basado en AspectJ, en el que un LAED se implementa como un *plugin* generando código AspectJ; la propuesta de Brichau et al. [45], cuyo objetivo es la construcción lenguajes específicos de aspectos que se puedan componer utilizando metaprogramación; Pluggable AOP [220], que se centra en cómo se componen dos o más LAEDs; y, Reflex [370], una implementación Java para un *kernel* versátil, multilenguaje y orientado a aspectos que soporta la semántica básica de varios lenguajes orientados a aspectos a través de modelos estructurales y de

comportamiento.

3.3.4 Separación multidimensional de competencias

La separación multidimensional de competencias está relacionada con la descomposición del software de acuerdo a las múltiples dimensiones de una competencia. Una dimensión de una competencia es un tipo de un elemento de interés para el sistema. Por ejemplo, en los lenguajes orientados a objetos, el interés se centra en los datos o en las clases; cada elemento de interés en esta dimensión se encapsula en una clase. Puede haber otras dimensiones como los aspectos [211], como el control de concurrencia; las características [384], como la persistencia; los roles [14] o los puntos de vista (*viewpoints*) [278]. La separación multidimensional de competencias pretende resolver el problema de los actuales lenguajes de programación conocido como la "tiranía de la descomposición dominante" [372], ya que solamente permiten la separación y encapsulación de una sola incumbencia a la vez, es decir, en una sola dimensión. Por ejemplo, en los lenguajes orientados a objetos la descomposición dominante son las clases; en los lenguajes funcionales, las funciones; o en los sistemas basados en reglas, las reglas. No pudiéndose, además, encapsular y manipular características en los lenguajes orientados a objetos, u objetos en los sistemas basados en reglas.

Los intereses o competencias son la motivación principal para organizar y descomponer el software en partes manejables y comprensibles. En cada etapa del ciclo de vida, y dependiendo del desarrollador concreto y del tipo del mismo, pueden resultar relevantes diferentes aspectos del sistema software a desarrollar. La *Separación Multidimensional de Competencias* [294] permite la separación simultánea de acuerdo a múltiples tipos arbitrarios de (dimensiones de) conceptos. Las competencias pueden solaparse e interactuar.

La separación multidimensional de competencias denota la separación de competencias que involucran las múltiples dimensiones de un concepto; la separación a lo largo de estas dimensiones de forma simultánea; la habilidad de manejar nuevas competencias y nuevas dimensiones del mismo concepto de forma dinámica, es decir, de manejarlos en el momento del ciclo de vida en el que aparezcan; y finalmente, competencias que se solapan o interactúan.

La propuesta para tratar con este tipo de separación proviene de la programación orientada a sujetos (*Subject-oriented programming*) [174] y se conoce como *hiperespacios* (en inglés, *hyperspaces*) [295]. Los hiperespacios permiten la identificación explícita de los elementos de interés del sistema, su encapsulación, la identificación y gestión de las relaciones entre los mismos, y su posterior integración. HyperJ^{TM} [71, 294], es la herramienta que da soporte a los hiperespacios para Java^{TM}.

La idea que subyace en este enfoque es el manejo de diferentes perspectivas de los objetos que van a ser modelados. La principal diferencia entre esta propuesta y la `POA` es que en la `SMC` se trabaja con varios modelos, y después se integran. Esta visión implica que todos los conceptos tienen la misma importancia. Por otra parte, la `POA` comienza

Término	Descripción
Unidad	Una unidad es un constructor sintáctico que se utiliza en el lenguaje con el que se construye el software. Así, si el software se construye en Java, las unidades serán clase, interfaz, método y atributo. Las unidades puede ser *primitivas* o *compuestas*. Una unidad primitiva se trata de forma atómica, mientras que una unidad compuesta agrupa a otras unidades. Así, por ejemplo, un método sería una unidad primitiva, y una clase sería una unidad compuesta.
Concern	Un *concern* es un elemento de interés para un sistema. Si pensáramos en un programa orientado a objetos, los *concerns* serían las clases (los elementos en los que se piensa a la hora de describir el sistema). Si el programa fuera un programa funcional, los *concerns* serían las funciones.
Espacio de *concerns*	Es el conjunto de todas las unidades de un sistema software.
Hyperspace	Es el espacio de concerns estructurado para dar soporte a la separación multidimensional del competencias. Las unidades se organizan en una matriz multidimensional. Cada eje de la matriz representa una dimensión, y cada punto del eje un *concern* de esa dimensión.
Hyperslice	Un *hyperslice* es un conjunto de unidades que son completas declarativamente, es decir, que deben declarar todo aquello a lo que hacen referencia. El *hyperslice* no necesita proporcionar una declaración completa de todo, por ejemplo, puede declarar una función sin proporcionar su implementación. La completitud declarativa es una propiedad importante para eliminar el acoplamiento entre *hyperslices*.
Relaciones de integración	Es la forma de indicar cómo se combinan dos *hyperslices*. Hay dos tipos distintos de relaciones, las *sensibles al contexto* y las *insensibles al contexto*. Las relaciones insensibles al contexto son aquéllas que se dan entre las diferentes competencias, independientemente del contexto de trabajo, mientras que las sensibles al contexto pueden aparecer, o no, dependiendo del contexto.
Hypermodule	Un hipermódulo está formado por un conjunto de *hyperslices* que se van a integrar, y un conjunto de *relaciones de integración*, que especifican cómo están integrados los *hyperslices* y cómo se deben integrar.

Tabla 3.4: Conceptos manejados en Hyper/J

modelando la funcionalidad básica y luego la aumenta con los aspectos. Por lo tanto, se puede decir que la funcionalidad básica es más relevante que los aspectos.

Las diferentes perspectivas del sistema se conocen en esta propuesta como hiperespacios (*hyperspaces*) y se componen utilizando lo que se llaman reglas de composición. Los principales términos manejados en este lenguaje se representan en la tabla 3.4.

Para ilustrar estos conceptos, vamos a utilizar el diagrama de clases que se muestra en la figura 3.5. El diagrama de clases modela parcialmente un sistema encargado de evaluar expresiones numéricas (una tarea parecida a la que haría cualquier compilador o intérprete sencillo). La aplicación está compuesta por tres módulos. El primero se encarga de calcular y mostrar el resultado de la evaluación de una expresión. El segundo se encarga de mostrar en un dispositivo de salida una expresión de forma textual. El último comprueba la corrección sintáctica y semántica de la expresión a evaluar. Las clases del diagrama tienen métodos modificadores y observadores, y los métodos `evaluate()`, `display()` y `check()`.

El sistema se puede ver desde dos perspectivas distintas, por un lado, las clases, y por otro, podemos pensar en los módulos como distintas características del mismo, entendiendo por característica el concepto utilizado en las familias de productos [384]. En la figura 3.5, se muestra cómo las distintas características se esparcen por las clases del sistema. Cada característica está representada con un recuadro de distinto color

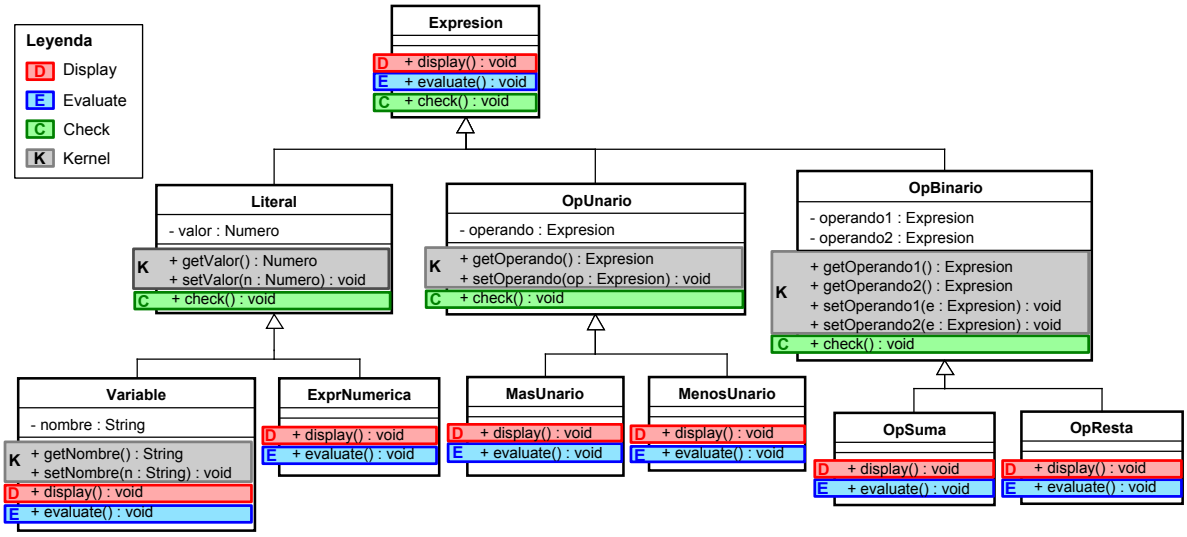


Figura 3.5: Diagrama parcial de clases para un sistema de evaluación de expresiones

que tiene la inicial de la característica (como indica la leyenda de la figura 3.5). Por ejemplo, los métodos relacionados con la característica `Evaluate` están en un recuadro marcado con la letra `E`. En la figura 3.6, se representa el hiperespacio de dos dimensiones correspondiente al sistema de evaluación de expresiones. El eje horizontal representa la dimensión del tipo de *concern* clase. Cada uno de los puntos sobre este eje representa a una de las clases del sistema. El eje vertical representa al tipo de *característica*, y cada unidad representa una característica del sistema, en este caso, `Evaluate`, `Check`, `Display` y, finalmente, `Kernel`. Cada unidad (en el ejemplo, las unidades se corresponden con los métodos declarados en las clases) está relacionada con un sólo *concern* en cada dimensión. Así, por ejemplo, el método `evaluate()` de la clase `Expresion` está relacionado con la clase `Expresion` de la dimensión de clases, y con la característica `Evaluate`, de la dimensión de características.

3.4 Modelado y diseño de software Orientado a Aspectos

Los sistemas orientados a aspectos se han de diseñar utilizando buenas prácticas en ingeniería del software. Así, teniendo en cuenta la importancia de las etapas de análisis y diseño en el desarrollo de cualquier proyecto, han surgido propuestas que van más allá de la separación de competencias a nivel de programación, y que se enfrentan a un desarrollo de software orientado a aspectos en las fases de análisis y diseño, para lo que se han acuñado los término *Análisis Orientado a Aspectos (`AOA`)* y *Diseño Orientado a Aspectos (`DOA`)* (también conocidos por su terminología inglesa *Aspect-*

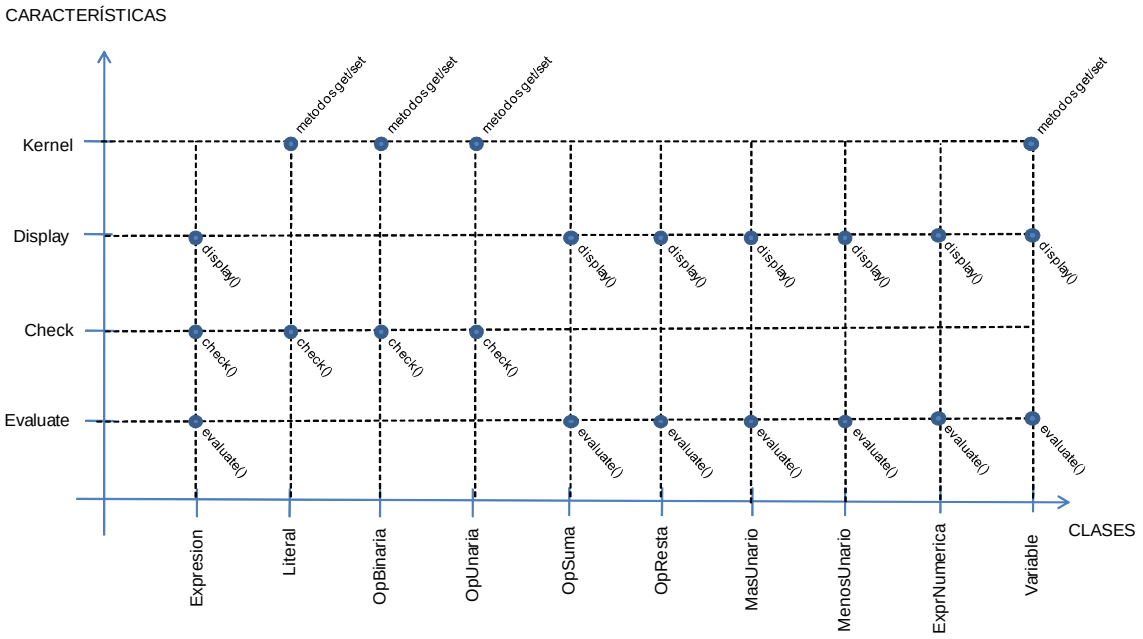


Figura 3.6: Hiperespacio bidimensional del sistema de evaluación de expresiones

Oriented Analysis(AOA) y *Aspect-Oriented Design (AOD)*, respectivamente).

Está claro que la separación de competencias es una característica común tanto a la programación orientada a aspectos como al modelado orientado a aspectos. Pero las diferencias entre los artefactos con los que se trabaja en cada uno (código y modelos, respectivamente), provoca también que existan técnicas diferentes en uno y en otro caso para conseguir esta separación. Inicialmente, a nivel de modelado, los desarrolladores utilizaban métodos y lenguajes orientados a objetos para diseñar aspectos (tales como `UML`). Pero el estándar `UML` no proporcionaba los constructores necesarios para describir aspectos. Así, era necesario un soporte especial para describir los aspectos. De esta forma, han surgido multitud de propuestas que tratan de resolver este problema (la tabla 3.5 muestra una relación de las más maduras aparecidas hasta la fecha). No se ha definido ningún estándar, aunque hay algunas propuestas más maduras que otras. Para clasificar y recoger las aportaciones y los valores añadidos de estas propuestas, en la bibliografía se han publicado varios estudios comparativos [41, 332, 59, 289, 348].

En la tabla 3.5 se recogen las propuestas analizadas en estos estudios comparativos, así como propuestas que han surgido con posterioridad a estas comparativas. La primera columna recoge el nombre de la propuesta en forma de acrónimo. La mayoría de estos acrónimos se han obtenido de [59], que es la comparativa que evalúa un mayor número de propuestas (un total de 22). En la segunda columna, aparece el nombre del principal autor de la propuesta. La tercera representa la referencia que explica mejor o de forma más completa la propuesta. A continuación, se muestran el año en que aparece la

primera referencia bibliográfica en la que se describe la propuesta, y el año de la última. Estas dos columnas dan una idea tanto de la madurez de la propuesta como de la vigencia de la misma. Por último, se muestran las distintas comparativas ordenadas cronológicamente. En cada fila, se representa una propuesta. Un *tick* (✓) en una celda significa que la propuesta se ha evaluado en la comparativa que representa la columna correspondiente. El símbolo $\simeq$ se utiliza debido a que, aunque la comparativa [348] ha considerado 14 propuestas suficientemente maduras, solamente ha evaluado 8. Así que el símbolo $\simeq$ en una celda significa que la propuesta se ha considerado, pero no se ha evaluado en la comparativa. Note también que hay propuestas que no se han evaluado en ninguna comparativa, bien porque aparecieron con posterioridad a las mismas, bien porque el sesgo de orientación a aspectos se le dió a la propuesta con posterioridad a las comparativas. Un ejemplo de este último caso es la propuesta denominada Protocol Modeling, ya que sus primeras publicaciones fueron en 2003, aunque su aplicación al campo de la orientación a aspectos no ha sido hasta 2010.

Las propuestas más antiguas relacionadas con el modelado orientado a aspectos se pueden agrupar en grandes familias. Por una parte, aquellas que intentan modelar los programas que se han implementado en alguna plataforma orientada a aspectos. Por otra parte, aquellas centradas en estructurar los modelos de acuerdo a los diferentes aspectos que los componen.

Las propuestas de la primera familia se centran en definir constructores que permitan expresar en un modelo los elementos nuevos introducidos por la programación orientada a aspectos, tales como los aspectos o los puntos de enlace. Su objetivo, no es, por tanto, estructurar los modelos, sino hacer que éstos puedan representar los detalles técnicos definidos en los lenguajes de programación orientados a aspectos. En este grupo se encuadran la mayoría de propuestas que aparecen en el grupo de etiquetado como Metamodelo de la tabla 3.5, que definen estos constructores utilizando técnicas de metamodelado, o en el grupo Profile, que extienden Unified Modeling Language (UML) con estos nuevos constructores.

Acrónimo	Autores	Referencia	Año primera ref.	Año última ref.	Blair et al. [Oct, 2004]	Reina et al. [Oct, 2004]	Chitchyan et al. [May, 2005]	Op de beeck et al. [Feb, 2006]	Schauerhuber et al. [Apr, 2007]
Estereotipo	J. Suzuki et al.	[368]	1999	1999	✓	✓	✓		
	J. L. Herrero et al.	[182]	2000	2002	✓		✓		
Metamodelo	C. v. F. García Chavez et al.	[395]	1999	2002		✓	✓		$\simeq$
	Y. Han et al.	[173]	2004	2005			✓		
	I. Philippow et al.	[303]	2003	2003		✓	✓		
	J. M. Lions et al.	[236]	2002	2002		✓			

Tabla 3.5: Propuestas de modelado orientado a aspectos.

		Referencia	Año primera ref.	Año última ref.	Blair et al. [Oct, 2004]	Reina et al. [Oct, 2004]	Chitchyan et al. [May, 2005]	Op de beeck et al. [Feb, 2006]	Schauerhuber et al. [Apr, 2007]
AOCE	J. Grundy	[165]	1999	2006	✓		✓	✓	≃
UMLAUT	W. M. Ho et al.	[184]	1999	2002	✓				≃
CAM/DAOP	M. Pinto et al.	[304]	2001	2005			✓	✓	≃
AOP2UML	M. M. Kandé et al.	[208]	2001	2002			✓		
Profile	A. A. Zakaria et al.	[416]	2002	2002		✓			
	M. Basch et al.	[28]	2003	2002		✓			
	J. Evermann	[111]	2007	2007					
	J. Uetanabara et al.	[205]	2009	2010					
ADM	J. P. Barros et al.	[27]	2002	2003			✓	✓	
AODM	D. Stein et al.	[364]	2002	2006	✓	✓	✓	✓	✓
Theme/UML	S. Clarke et al.	[60]	2002	2011	✓	✓	✓	✓	✓
SUP	O. Aldawud et al.	[8]	2002	2005		✓	✓	✓	✓
AAM	R. France et al.	[131]	2002	2007			✓	✓	✓
CoCompose	D. Wagelaar et al.	[402]	2002	2006			✓	✓	
UFA	S. Herrmann	[183]	2002	2002		✓	✓	✓	
UML4AO	R. Pawlak et al.	[302]	2002	2005			✓	✓	✓
AVA	M. Katara et al.	[210]	2003	2006	✓		✓	✓	≃
AML	I. Groher et al.	[155]	2003	2004		✓	✓	✓	
Stratified Frameworks	T. Kühne et al.	[21]	2003	2006			✓	✓	✓
AOSDUC	I. Jacobson	[198]	2003	2007			✓		✓
MDA	S. J. Mellor	[246]	2003	2003		✓			
	V. Kulkarni et al.	[227]	2003	2003		✓			
AODM	J. Gray et al.	[150]	2003	2011					
Protocol Modeling	A. McNeile et al.	[243]	2003	2010					
AOMDF	R. France et al.	[131]	2004	2006			✓	✓	✓
VC	A. Muller	[263]	2004	2006			✓	✓	
IDAM	W. Coelho et al.	[63]	2004	2006			✓		≃
AOSF	M. Mahoney et al.	[240]	2004	2010					
JPDD	D. Stein et al.	[365]	2004	2011					
MWACSL	A. M. Reina et al.	[332]	2004	2010			✓		
UML Bottom Up	M. Tkatchenko et al.	[381]	2005	2005					
MWEAVR	T. Cottenier et al.	[73]	2005	2007					✓
SBWS	J. Klein et al.	[214]	2005	2007					✓
MATA	J. Whittle et al.	[406]	2007	2009					
RAM	J. Kienzle et al.	[212]	2007	2010					
HiLa	J. Zhang et al.	[417]	2007	2011					
AO Executable UML 2.0	L. Fuentes et al.	[134]	2007	2009					
Smart Adapters	B. Morin et al.	[230]	2007	2010					
GReCCo	A. Hovsepyan et al.	[189]	2008	2009					
KerTheme	O. Barais et al.	[23]	2008	2010					
GeKo	B. Morin et al.	[262]	2008	2010					
AOM-DSML	A. Hovsepyan et al.	[190]	2009	2010					
Aspect Weaver UML	M. Nouh et al.	[276]	2009	2010					

Tabla 3.5: Propuestas de modelado orientado a aspectos.

La segunda familia de propuestas se centra en estructurar los modelos de acuerdo a los diferentes aspectos que los componen. Estas propuestas se quedan a nivel de modelado y en ellas no aparecen los constructores de los lenguajes de programación orientados a aspectos. A este grupo pertenecen propuestas como Theme/UML o AAM (Ver tabla 3.5).

Más recientemente, la aparición del DSDM y de MDA ha causado que exista un grupo de propuestas que abogan por mantener una separación horizontal de conceptos aplicando una filosofía de modelado orientado a aspectos y una separación vertical usando la arquitectura de modelos propuesta en MDA. Algunas de las propuestas de este grupo son una evolución de algunas de las propuestas más antiguas recogidas en las dos familias mencionadas anteriormente. Por ejemplo, AOMDF es una evolución de AAM.

En relación con el DSDM también han surgido propuestas como la de Gray o AOM-DSML que abogan por separar los conceptos mediante lenguajes de modelado específicos de dominio.

Al igual que en las propuestas a nivel de programación, en diseño también existen las aproximaciones simétricas y asimétricas. Como no es el objetivo de esta tesis evaluar todas y cada una de las propuestas, en los siguientes subapartados se detallarán dos propuestas, una por cada aproximación. Se han escogido las dos propuestas que se consideran más maduras. Por una parte, y como representante del grupo de propuestas simétricas, en el apartado 3.4.1 se describe las líneas generales de Theme/UML, mientras que como representante de las propuestas asimétricas, se detalla AAM en la sección 3.4.2

3.4.1 Theme/UML

El enfoque Theme [60] cubre tanto la etapa de análisis como la de diseño, aunque este apartado solamente se centrará en la última. Theme/Doc es la parte de la propuesta que trata con los requisitos y, principalmente, consiste en un conjunto de heurísticas para analizar la documentación de requisitos y encontrar posibles temas. El *tema* (*theme*) es el elemento central de esta propuesta. Está más cercano al concepto o incumbencia de las propuestas simétricas que al aspecto de las propuestas asimétricas. Un tema, según las autoras, se define como la encapsulación de una incumbencia (o *concern*). Por otra parte, Theme/UML se encarga de definir los conceptos en diseño, tomando como punto de partida los resultados obtenidos en la etapa de análisis con Theme/Doc. Es decir, Theme/UML parte del conjunto de temas obtenidos en la etapa de requisitos.

Por cada tema identificado se definirá un modelo de diseño. Un sistema, por tanto, estará compuesto por un conjunto de temas, y el diseño del mismo estará formado por un conjunto de modelos definidos de forma totalmente separada unos de otros. Se puede considerar que en Theme hay dos tipos de temas, los que se conocen como *temas base*, y los *temas aspecto*. Un tema base se especifica por completo en UML, y se representa por un paquete con el estereotipo `<<theme>>`, que contendrá un conjunto de diagramas UML, tanto estructurales como de comportamiento. Si volvemos al ejemplo del sistema de evaluación de expresiones introducido en la sección 3.3.4, se podría definir un tema por cada uno de las incumbencias (`Kernel`, `Display`, `Evaluate` y `Check`). Las figuras

3.7 y 3.8 muestran, respectivamente, el diseño de dos temas base (`kernel` y `evaluate`). Como se puede comprobar, en cada tema solamente intervienen aquellos elementos relacionados con la incumbencia del mismo.

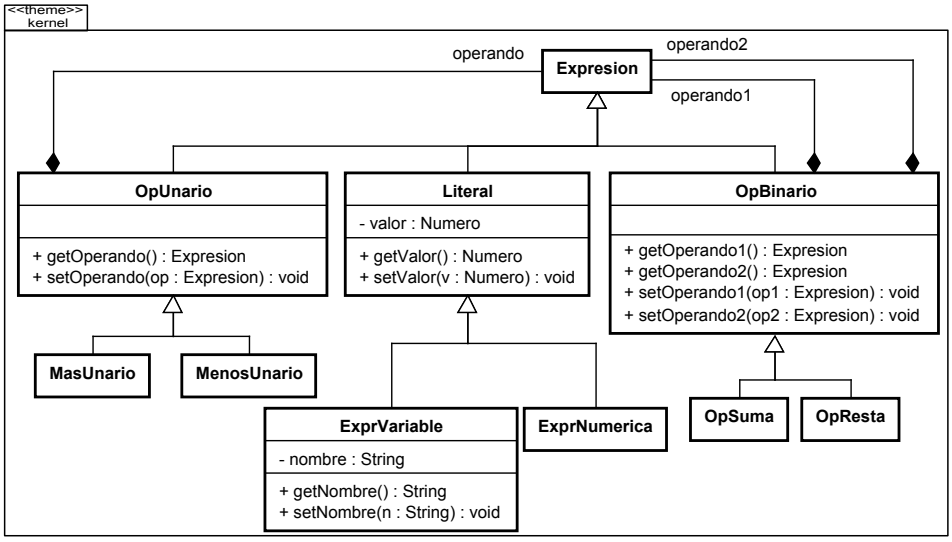


Figura 3.7: Tema de tipo base para la característica kernel (adaptado de [60])

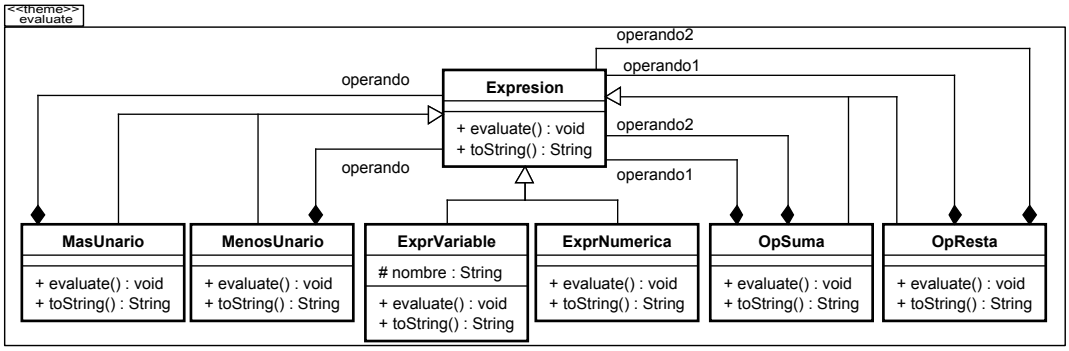


Figura 3.8: Tema de tipo base para la característica evaluate (adaptado de [60])

Los temas de tipo aspecto son aquellos que pueden definir un comportamiento que se ejecuta cuando tiene lugar algún otro comportamiento definido en otro tema. Así, estos temas no se especifican por completo en `UML` estándar, ya que utilizan una notación propia para parametrizar el comportamiento que puede ser ejecutado a partir de un tema base. Al igual que los temas base, los temas de tipo aspecto se especifican en un paquete con el estereotipo `<<theme>>`. También pueden incluir diagramas `UML` estructurales y de comportamiento, la diferencia reside en que al paquete se le asocia

una plantilla que define los puntos desde los que se puede lanzar el comportamiento. Si quisiéramos trazar la ejecución de algunos de los métodos de nuestro sistema de evaluación de expresiones, podríamos definir un tema de tipo aspecto como el que se muestra en la figura 3.9, donde puede verse un tema `logging` que contiene dos diagramas, uno estructural (el de las clases `ClaseTrazada` y `FicheroTraza`), y el otro, de comportamiento (el diagrama de secuencia). La parametrización se representa en un recuadro dibujado con línea discontinua situado en la esquina superior derecha del paquete. En el ejemplo de la figura 3.9, el parámetro es un método (`opATrazar`) de una clase (`ClaseTrazada`). El método que se quiera registrar, hará el papel del `opATrazar` en el diagrama de secuencia.

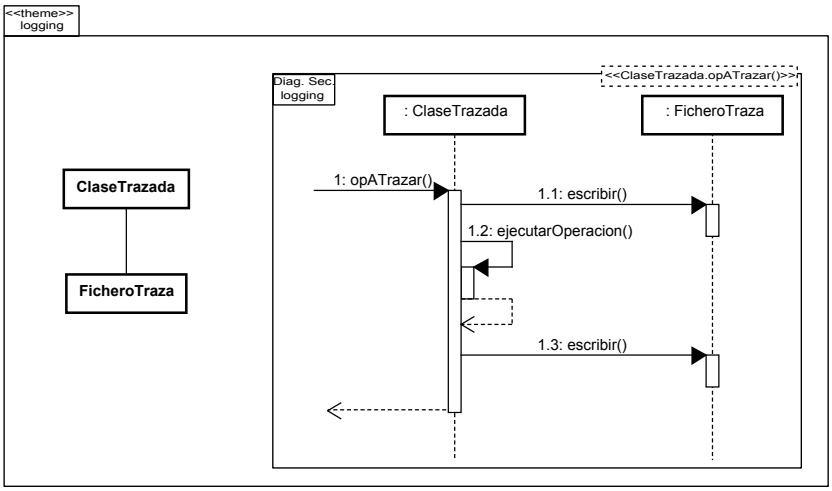


Figura 3.9: Tema de tipo aspecto para la traza (adaptado de [60])

Además de diseñar cada uno de los temas, hay que especificar cómo se relacionan unos con otros. En Theme, existen dos tipos de relaciones, la de aquellos temas que comparten conceptos del dominio (una relación de tipo *match*), y la de aquéllos que se entrememezclan (relación de tipo *crosscut* o *bind*).

Dos temas comparten conceptos si tienen elementos de diseño que referencian a los mismos elementos del dominio. Este tipo de relación se tiene en cuenta en los enfoques simétricos, pero no en los asimétricos. Un ejemplo de este tipo de relación se daría entre los temas `kernel` y `evaluate`. La figura 3.10 muestra la especificación de la relación entre estos dos temas, donde se indica que la integración se los dos temas se realizará por nombre.

El segundo tipo de relación implica que un comportamiento de un tema se dispara por comportamientos en otros temas. Se puede decir que este tipo de relación es la que se tiene en cuenta en los enfoques asimétricos. Es el tipo de relación que se da entre temas base y temas de tipo aspecto. Los temas base son los que disparan el comportamiento mientras que los temas aspectos son el comportamiento a ejecutar. Un ejemplo de este

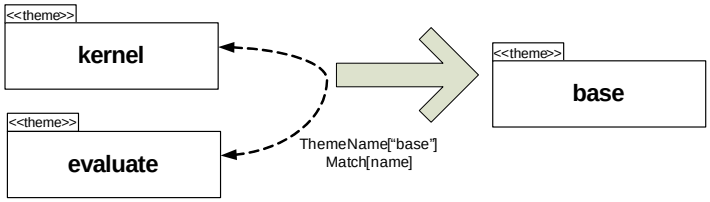


Figura 3.10: Especificación de relación de integración de conceptos en Theme

tipo de relación se representa en la figura 3.11. Es el que se da entre el tema `logging` y el resultado de componer los temas `kernel` y `evaluate`, que en la figura 3.10 se ha llamado tema `base`.

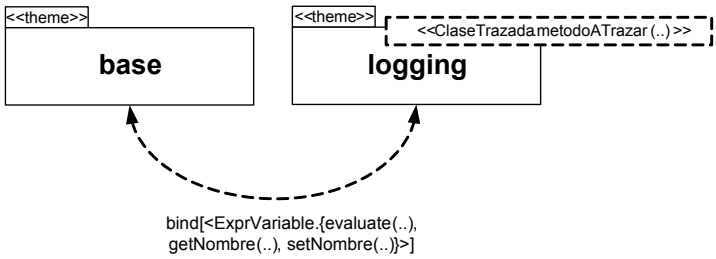


Figura 3.11: Especificación de relación tipo *binding* en Theme

Como se puede comprobar, Theme/UML se aleja un poco más del estándar `UML` a la hora de especificar las relaciones. Para indicar cómo un tema se relaciona con el resto de temas del sistema, Theme/UML ha definido un nuevo tipo de relación, llamada *relación de composición*, que permite identificar aquellas partes de un tema que se relacionan con otras, y que, por tanto, deberían componerse en algún momento. Esta relación deberá especificar cuándo y dónde debe ocurrir el comportamiento adicional en los temas, si se trata de una relación de tipo *binding*. Si, en cambio, hablamos de una relación de integración, entonces habría que identificar qué elementos del tema se corresponden con cuáles otros y la manera de integrarlos.

Finalmente, podemos decir que la aproximación Theme está más cercana a las aproximaciones simétricas. Los temas pueden verse como conceptos individuales, aunque los autores han adoptado alguna terminología de los enfoques asimétricos, tales como los términos corte (*crosscut*) y aspecto.

3.4.2 Aspect-Oriented Architecture Model (AAM)

La propuesta *Aspect-oriented Architecture Model* [131, 137] es una propuesta de modelado orientado a aspectos asimétrica, que propone *Modelo de Arquitectura Orientado a Aspectos (`MAA`)* para tratar con los *concerns* de manera independiente de la tecnología

utilizada. Un Modelo de Arquitectura Orientado a Aspectos (MAA) está compuesto por un conjunto de modelos de aspectos y un modelo de arquitectura base, llamado modelo primario.

Antes de que un modelo de aspectos se pueda componer con un modelo primario en un dominio de aplicación, el modelo del aspecto se debe instanciar en el dominio de la aplicación. La instanciación se consigue ligando los elementos del modelo de aspecto a elementos del dominio de la aplicación. El resultado de esta ligadura se llama *modelo de aspecto específico de contexto*. Finalmente, los modelos de aspectos específicos de contexto y el modelo primario se componen para obtener una vista de diseño integrada.

Resumiendo, se puede decir que esta propuesta da soporte, por una parte, a la descripción de *crosscutting concerns* como vistas de modelado, llamadas aspectos. Por otra, a la síntesis e integración de modelos mediante la composición de las vistas del modelo primario y los aspectos. Y, por último, a la identificación y resolución de conflictos y propiedades emergentes no deseables que pueden aparecer como resultado de la integración de los modelos de aspectos y primario.

En esta propuesta, un modelo de aspecto es un patrón que caracteriza a una familia lógica de soluciones para el concepto que modela el aspecto. Los patrones se describen utilizando plantillas UML parecidas a las que se usan en Theme/UML (ver sección 3.4.1). La notación de plantillas utilizada está basada en el lenguaje *Role-Based Metamodeling Language (RBML)* [130]. Para componer un modelo de aspectos con un modelo primario, hace falta que previamente se instancie el aspecto, es decir, hay que ligar los parámetros definidos en el aspecto a los valores específicos de la aplicación.

Con respecto a la composición de los modelos de aspectos y el modelo primario, los autores han optado por una solución en la que es necesaria la intervención parcial del usuario, de tal forma que hay un procedimiento de composición definido por defecto, pero éste se puede cambiar utilizando directivas de composición. En la figura 3.12 se muestra una visión general de los pasos a dar en esta composición.

Así, podemos decir que en esta propuesta, el diseño de una aplicación se expresa utilizando los siguientes artefactos:

1. Un *modelo primario*, que describe la lógica de negocio de la aplicación. Estará formado por una serie de diagramas UML que describen una vista de la arquitectura base de la aplicación.

2. Un *conjunto de modelos de aspectos genéricos*, donde cada modelo es una descripción genérica de un concepto. Los autores solamente tratan con conceptos o competencias concretas, en el sentido de que éstas se puede expresar en términos estructurales y funcionales en un modelo (como contraposición a las competencias concretas están los conceptos o competencias cualitativas, que están basadas en las cualidades o atributos de un sistema, y que no se reflejan términos estructurales y funcionales en los modelos). Los aspectos genéricos se modelan, por tanto, con plantillas UML.

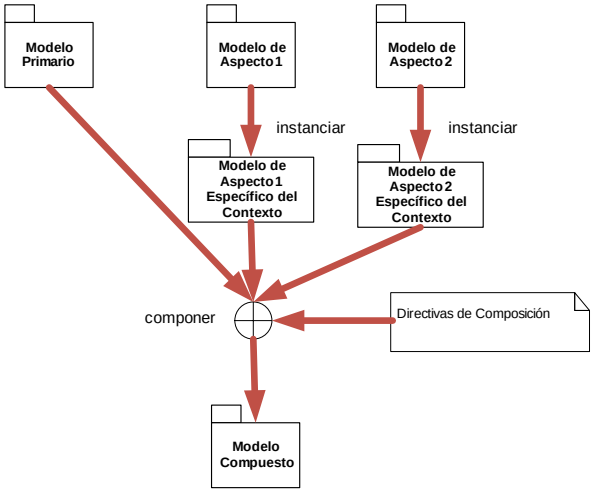


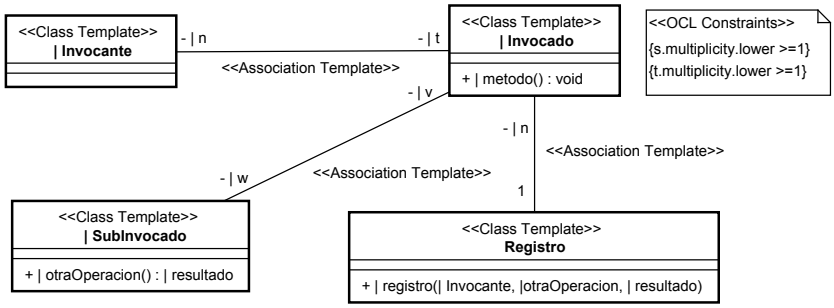
Figura 3.12: Visión general de la fase de composición propuesta en AAM

3. Un *conjunto de ligaduras*, que determinan en qué parte del modelo primario se han de componer los aspectos, o lo que es lo mismo con qué valores específicos del dominio de la aplicación se instancian los aspectos.

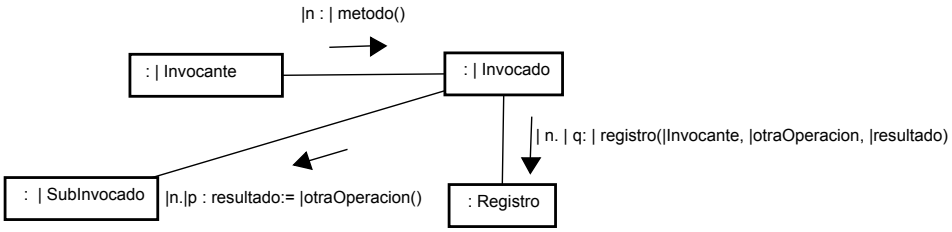
4. Un *conjunto de directivas de composición* que influencian en cómo los modelos de aspectos se componen con el modelo primario.

Los modelos de aspectos se representan utilizando paquetes parametrizados, pero en lugar de utilizar la notación `UML` en la que se listan todos los parámetros en la cabecera del paquete, los autores han optado por marcar los parámetros directamente en las plantillas del modelo utilizando el símbolo '|' delante del mismo. En la figura 3.13 se muestran dos diagramas correspondientes a una plantilla para el aspecto de *logging*. Se puede ver cómo con dos diagramas, uno estático (figura 3.13(a)) y uno de comportamiento (figura 3.13(b)), se modela el aspecto. Además, cada una de las etiquetas que comienzan con | representa un parámetro de la plantilla. Por ejemplo, la clase `|Invocante` representa a la clase que invoca al método que se desea registrar. La clase `|Invocado` representa a la clase que contiene el método a registrar.

La composición de este aspecto con un modelo primario implica, en primer lugar, instanciar el modelo de aspectos utilizando ligaduras para producir un modelo de aspectos específico de contexto, y, en segundo lugar, integrar el modelo de aspecto específico de contexto con el modelo primario. Las ligaduras se utilizan para relacionar un elemento del modelo de aspectos con un elemento del modelo, y se suele expresar como un par de la forma `(nombre de elemento del aspecto, nombre de elemento del modelo)`. El nombre del elemento del modelo puede ser tanto un elemento del modelo primario, como un elemento específico de la aplicación que se añadirá al modelo durante la composición. Por ejemplo, si se quiere registrar el resultado de ejecutar un método



(a) Plantilla de diagrama de clases para el aspecto logging



(b) Plantilla de diagrama de colaboración para el aspecto logging

Figura 3.13: Modelo para el aspecto logging

llamado `calcularResultado`, se puede definir una ligadura del tipo `(otraOperacion, calcularResultado)`.

Para realizar la composición, se utiliza un procedimiento de composición por nombre, de forma que aquellos elementos que tienen el mismo nombre se mezclan para formar un elemento simple en el diagrama resultado de la composición. De cualquier manera, se pueden definir otras directivas de composición. Si hay varios aspectos a componer, con las directivas de composición se puede: determinar el orden en el que se van componiendo estos aspectos con el modelo primario; definir la precedencia o las relaciones de anulación entre elementos que concuerdan en el modelo primario y de aspectos y que tienen propiedades que entran en conflicto; determinar los elementos que se renombran (por ejemplo, para resolver conflictos). Es decir, las directivas de composición van a permitir variar la forma en que el modelo primario y los aspectos se componen.

3.5 Desarrollo de Software Orientado a Aspectos en las fases tempranas

De igual forma que ocurría con el diseño orientado a aspectos, surgen problemas parecidos a la hora de analizar los requisitos de un sistema para determinar cómo se va a diseñar el mismo, teniendo como consecuencia que las técnicas de descomposición de requisitos desde una perspectiva orientada a objetos no son suficientes cuando se trata de sistemas orientados a aspectos. Intentando resolver esta problemática ha surgido el

análisis orientado a aspectos (AOA), y lo que en la bibliografía se conoce como aspectos tempranos (o en su terminología inglesa, *early aspects*).

Los aspectos también pueden aparecer en las fases iniciales del ciclo de vida del desarrollo de un proyecto software. Según [314], los *aspectos tempranos* se refieren a aquellas propiedades de tipo *crosscutting* que aparecen a nivel de requisitos y de arquitectura. Podemos decir que los aspectos se manifiestan en los requisitos como un comportamiento que se describe de tal forma que puede ser disparado por muchos otros comportamientos. Por ejemplo, dentro de los requisitos que describen un sistema bancario, la gestión de transacciones es un comportamiento necesario para un amplio número de actividades, como la transferencia de dinero entre cuentas, el cálculo de intereses, ...

Al igual que ocurre en la parte de modelado, como parte del proyecto AOSD Europe se ha realizado un estudio del estado del arte de las propuestas de separación de conceptos a nivel de análisis [59], algunas de las cuales como Theme/Doc [60], o AORE [164], tienen su continuación en diseño. Aunque existen también numerosas propuestas, no es el ámbito de esta tesis el trabajar a nivel de requisitos, así que se deja al lector que explore [59] o [314] para profundizar en las propuestas de esta etapa del ciclo de vida.

3.6 Sumario

El propósito de este capítulo era doble, por una parte, introducir al lector en la terminología y conceptos utilizados en el nuevo paradigma de desarrollo conocido como Desarrollo de Software Orientado a Aspectos; por otra, dar una visión general de las distintas técnicas de separación de conceptos a lo largo del ciclo de vida del mismo.

Las bases para el posterior desarrollo de este capítulo se sentaron en [317], que puede considerarse como una primera aproximación a la terminología y conceptos manejados en el área de la separación avanzada de conceptos. Las distintas aproximaciones para separar conceptos a nivel de implementación, se estudiaron en [324]. Finalmente, y como paso previo para alcanzar el objetivo del capítulo, las propuestas de modelado de aspectos se estudiaron en [332].

Modeling is the future ...
And the promise here is that you write a lot less code, that you have a model of the business process ...

Bill Gates

4

Desarrollo de Software Dirigido por Modelos

Este capítulo pretende dar una visión general de lo que se conoce como Ingeniería Dirigida por Modelos, centrándose sobre todo en la propuesta de la `OMG`*, conocida como* `MDA`*, y en las técnicas y herramientas existentes para transformar y componer modelos. Para ello, el capítulo se ha estructurado de la siguiente forma: en primer lugar, en la sección 4.2 se introduce la terminología básica utilizada en este área de investigación. En la sección 4.3 se muestran los principios sobre los que se asienta el Desarrollo de Software Dirigido por Modelos. La sección 4.4 explora la relación entre los lenguajes específicos de dominio y los modelos. Una vez que se tiene una visión de conjunto, la sección 4.5 se centra en la propuesta de la* `OMG` *para enfrentarse al* `DSDM`*. La sección 4.6 da una visión general de las técnicas existentes para realizar transformaciones, tanto de modelo a texto como de modelo a modelo. La sección 4.7 se dedica a otra de las operaciones esenciales dentro del* `DSDM`*, la composición de modelos. Luego, en la sección 4.8 se exploran los estándares y las herramientas existentes para describir metamodelos, para transformar y para componer modelos. Finalmente, la sección 4.9 resume el capítulo.*

4.1 Introducción

Tradicionalmente los modelos se han utilizado en la ingeniería del software para definir y entender el dominio de un problema o los diferentes aspectos de la arquitectura de un sistema, permitiendo concentrarse en los detalles que son relevantes para ese problema concreto e ignorando los restantes. Una de las cuestiones que desmotivaban más a los

desarrolladores a la hora de utilizar modelos en la ingeniería del software es que una vez que se obtenía la implementación de un sistema a partir de una serie de modelos, éstos no se actualizaban para reflejar las decisiones tomadas en la implementación.

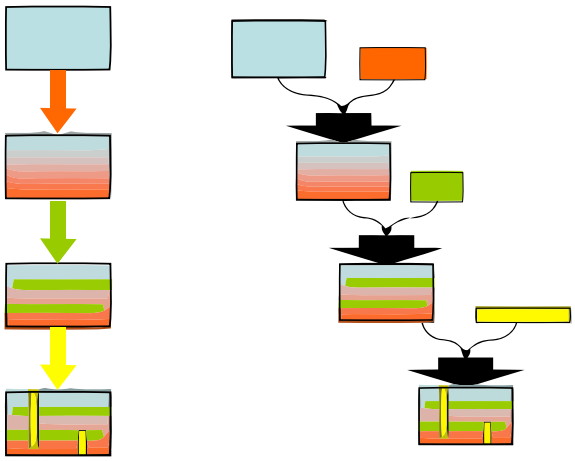
Para resolver este problema, entre otros, ha aparecido la *Ingeniería Dirigida por Modelos (IDM)*, o en inglés *Model-Driven Engineering (MDE)* [353], que aboga por el uso sistemático de los modelos como principal artefacto de desarrollo durante todo el ciclo de vida de un proyecto software. Un término que también se utiliza para referirse a este nuevo paradigma es DSDM [47], en inglés, *Model-Driven Software Development (MDSD)*. La aplicación de esta filosofía ha provocado que se incorporen como parte central de proceso del desarrollo las ideas de modelo, modelado y transformación entre modelos. Un modelo, por tanto, se utiliza para describir una parte de un sistema software, permitiendo incrementar el nivel de abstracción con el que se trabaja, y pudiendo ser usado para describir tanto el dominio del problema como el dominio de la solución.

Para describir algo mediante un modelo se necesita un lenguaje de modelado, que puede considerarse como una notación formal que permite la manipulación automática o semi automática de modelos. Del mismo modo, para diseñar los lenguajes de modelado se utilizan los lenguajes de metamodelado. Un *metamodelo* se utiliza para describir la sintaxis abstracta de un lenguaje de modelado.

Las relaciones entre modelos se pueden definir mediante transformaciones o mediante modelos de composición. Una *transformación de modelos* toma una serie de modelos como entrada y produce una serie de modelos como salida, mientras que la *composición de modelos* puede verse como un tipo especial de transformación de modelos que toma como entrada dos (o más) modelos, *Ma* y *Mb*, y combina sus elementos en un modelo de salida *Mab* [85, 114]. Tanto la transformación como la composición de modelos son operaciones fundamentales en la Ingeniería Dirigida por Modelos y persiguen un mismo objetivo [30], incorporar nuevas características a un modelo. La principal diferencia entre ellas es que en la composición de modelos, las nuevas características a incorporar al modelo se hacen explícitas, ya que están descritas en uno o más modelos fuentes, mientras que en la transformación de modelos las nuevas características están definidas de forma implícita en las acciones de transformación que se ejecutan en el modelo de entrada. La figura 4.1 refleja esta idea. De esta forma, J.M. Jézéquel representa cómo las características a incorporar al modelo vienen dadas en las transformaciones (figura 4.1(a)), mientras que en la figura 4.1(b) estas características están definidas de forma explícita en otro modelo.

4.2 Terminología

Los modelos se han convertido en el artefacto principal de desarrollo del software dentro del DSDM. Si buscamos en el diccionario de la Real Academia de la Lengua Española [316] la entrada para la palabra modelo, veremos que existen varias definiciones. La que mejor se ajusta al ámbito de la ingeniería del software es: *"Esquema teórico, generalmente*



(a) Esquema de una transformación de modelo.

(b) Esquema de una composición de modelos

Figura 4.1: Transformación de modelos vs. composición de modelos, según Jézéquel

en forma matemática, de un sistema o de una realidad compleja, como la evolución económica de un país, que se elabora para facilitar su comprensión y el estudio de su comportamiento". Se puede decir, por tanto, que un modelo es una representación de una parte de un sistema, que permite incrementar el nivel de abstracción con el que se trabaja, y facilita así su comprensión. El grado de complejidad del mismo dependerá del nivel de detalle con el que se describa el modelo.

Para describir algo mediante un modelo se necesita un lenguaje de modelado. Un lenguaje de modelado puede considerarse como una notación formal que permite la manipulación automática o semi-automática de modelos. Un ejemplo de este tipo de lenguajes es `UML` [284]. Se dice, por tanto, que un modelo es conforme a un lenguaje de modelado. Es decir, el lenguaje de modelado restringe de alguna manera los constructores o elementos que se pueden utilizar en el modelo. Para diseñar estos lenguajes de modelado se utilizan los lenguajes de metamodelado. Un *metamodelo* se utiliza para describir la sintaxis abstracta de un lenguaje de modelado. Se puede decir que en él se indican los conceptos que pueden usarse en un modelo y la relación entre los mismos. Un ejemplo de lenguaje de metamodelado es `MOF` [286].

Así, un lenguaje de metamodelado no es más que un tipo especial de lenguaje de modelado que se utiliza para describir modelos. Por lo tanto, como tal lenguaje de modelado, también tiene que ser conforme a un metamodelo, que se conoce con el nombre de *meta-metamodelo*. El meta-metamodelo se utiliza para definirse a sí mismo. Para recoger esta relación entre lenguajes de modelado, la `OMG` ha definido una arquitectura de cuatro niveles, a los que ha denominado M0, M1, M2 y M3. Para cada nivel, propone

un estándar de lenguaje de modelado. En la figura 4.2 se muestra un esquema de estos niveles, las relaciones entre ellos y los estándares propuestos por la OMG.

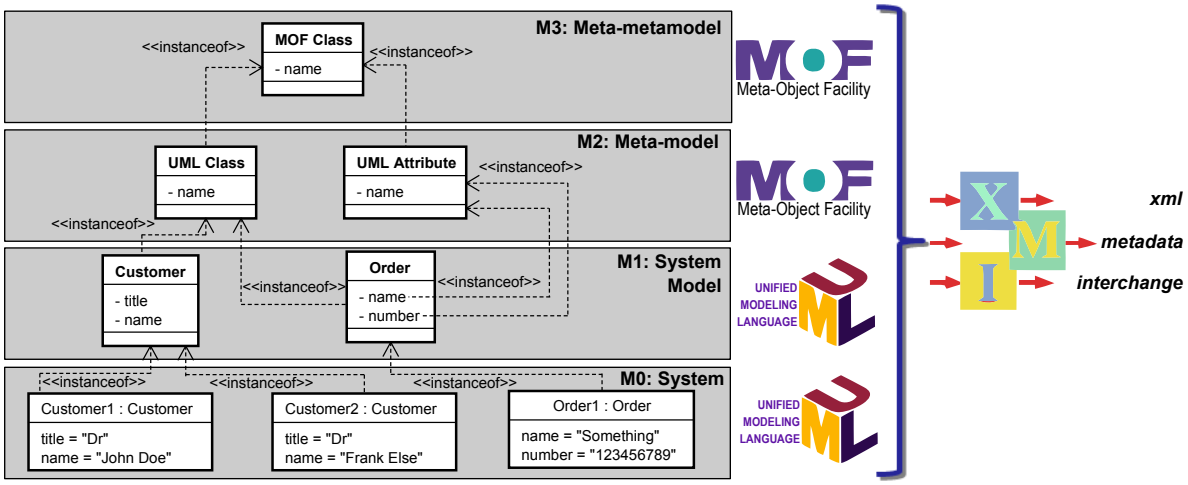


Figura 4.2: La arquitectura de cuatro capas de modelos propuesta por la OMG

En la parte alta de la figura 4.2, en el nivel M3, están los lenguajes de metamodelado. El estándar propuesto por la OMG para este nivel es el llamado *Meta Object Framework* (MOF) [286]. Los lenguajes de este nivel se utilizan para definirse a sí mismos, es decir, MOF está definido en MOF, lo que implica que por encima del nivel M3 no hay otro nivel, sino que hay una relación reflexiva del nivel M3 consigo mismo. En el nivel M2 se encuentran situados los lenguajes de modelado como UML [284]. Para describir la sintaxis abstracta de UML se utiliza MOF. El modelo de un sistema concreto se encuentra en el nivel M1. Para describir el sistema, uno de los lenguajes propuestos por la OMG es UML. El modelo del sistema se puede ver como una instancia del metamodelo de UML, en el caso de la figura 4.2. Finalmente, en el nivel M0 se sitúa el sistema concreto, una instanciación del modelo situado en el nivel M1.

Como formato estándar de intercambio de estos modelos entre herramientas, la OMG propone el estándar *XML Metadata Interchange (XMI)* [287]. En el nivel M0 de la figura 4.2, se puede ver un sistema en el que hay dos clientes (representados como instancias de la clase `Customer`) y un pedido (un objeto instancia de la clase `Order`). En el nivel M1 se encuentran las clases `Customer` y `Order`, que son instancias de la metaclase `UMLClass`. Además, cada clase tiene definidos dos atributos (`title` y `name`, en el caso de `Customer`, y `name` y `number` en el caso de `Order`). Estos atributos también son instancias de una metaclase, la llamada `UMLAttribute`. En el nivel M2, se encuentran las metaclases `UMLClass` y `UMLAttribute`, que son instancias de la meta-metaclase `MOFClass` situada en el nivel M3. Finalmente, hay que notar que para facilitar la comprensión de la arquitectura, en el diagrama de la figura 4.2 no se han representado las relaciones entre clases, como por ejemplo, la relación existente entre las clases `Customer`

y `Order`.

Además de los modelos, otra de las ideas clave que se ha introducido en el `DSDM` es la de transformación entre modelos. Una *transformación de modelos* toma un conjunto de modelos como entrada y produce un conjunto de modelos como salida. Los principales elementos involucrados en una transformación de modelos son: un modelo origen o de entrada, un modelo destino o de salida (ambos conformes a sus respectivos metamodelos); una definición de transformación, que será descrita en un lenguaje, y que se ejecutará en base a los metamodelos origen y destino. El lenguaje en el que se describe la transformación también debe ser conforme a un metamodelo. La figura 4.3 muestra un esquema los elementos que intervienen en una transformación de modelos y su relación. El artefacto encargado de ejecutar la transformación se conoce como motor de transformación.

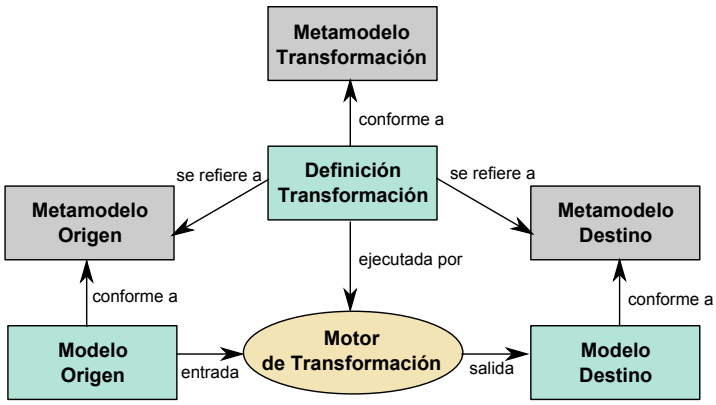


Figura 4.3: Elementos que intervienen en una transformación de modelos

Según France y Bieman [129], las transformaciones se pueden clasificar en transformaciones verticales y transformaciones horizontales. Las transformaciones verticales son aquellas que se aplican a un modelo para obtener otro expresado con un nivel de abstracción diferente, mientras que las transformaciones horizontales se aplican a un modelo para obtener otro expresado en el mismo nivel de abstracción. Un ejemplo de transformación horizontal puede ser una refactorización, en la que lo que se hace es modificar la estructura del modelo de entrada, pero el modelo de salida está expresado en el mismo nivel de abstracción. Un ejemplo de transformación vertical puede ser un refinamiento. Además de estos dos tipos de transformaciones, Czarnecki [77] define las transformaciones oblicuas como aquellas que tienen componentes tanto verticales como horizontales.

Además de esta clasificación, podemos decir que hay *transformaciones modelo a modelo (M2M)* que son aquellas transformaciones que toman uno o varios modelos como entrada y devuelven uno o varios modelos como salida, y *transformaciones de modelo a texto (M2T)*, que toman uno o varios modelos como entrada y producen uno

o varios artefactos textuales como salida.

Junto con la transformación de modelos, la *composición de modelos* es otra operación esencial dentro de la Ingeniería Dirigida por Modelos que supone combinar varios modelos. La composición de modelos es una operación que se utiliza en diversas áreas de investigación como el modelado orientado a aspectos, la integración de esquemas de bases de datos, o la transformación de modelos, lo que ha dado lugar a diferentes terminologías, que pueden llegar a causar confusión. En este sentido, han surgido algunos trabajos que intentan dar un conjunto de definiciones comunes, como [51] o [393], y otros en los que se pretende definir un marco que permita comparar diferentes procesos de composición, como [200]. Un esquema general para la composición de modelos es el que se presenta en la figura 4.4, definido en [199].

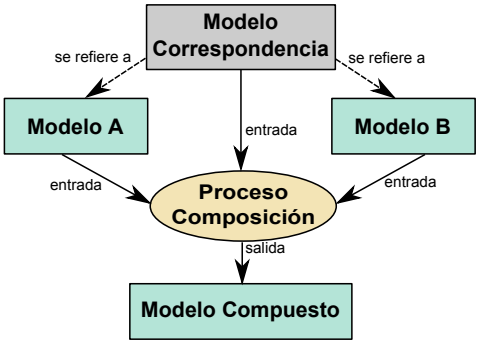


Figura 4.4: Esquema de composición de modelos

El *proceso de composición de modelos* [393] es el proceso de integrar una serie de modelos, bien asegurando su interoperabilidad como modelos autónomos, bien combinándolos en nuevos modelos compuestos. Es importante notar que la composición de modelos no siempre produce un modelo compuesto, sino que puede dejar que los modelos originales sean autónomos y asegurar su interoperabilidad. Esto es especialmente importante para sistemas complejos, ya que componer todos los modelos en uno no es deseable puesto que este modelo sería difícil de entender y poco manejable.

El proceso de composición toma como entrada dos modelos (`Modelo A` y `Modelo B`) y un modelo de correspondencia y devuelve un modelo compuesto. Los *modelos de entrada* son modelos sobre los que se hacen consultas en una actividad del proceso de composición. El *modelo de correspondencia* [51] es la forma de representar enlaces entre los modelos de entrada. Dependiendo del ámbito y de cómo se defina el proceso de composición, este modelo de correspondencia puede tener distintas formas, por ejemplo, en AMW [93], este modelo de correspondencia se implementa mediante una interfaz de usuario y se refleja en un modelo de *weaving*, mientras que en EML [97] las correspondencias se definen mediante reglas de comparación en ECL, un lenguaje de comparación de modelos. El `Modelo Compuesto` es un *modelo de salida* que se crea y/o actualiza en el proceso de composición.

Finalmente, como resumen de todos los conceptos vistos en esta sección, la figura 4.5 muestra un mapa conceptual en el que se recogen estos conceptos y sus relaciones. En el modelo conceptual se han añadido también algunos conceptos que se definirán en las siguientes secciones, pudiendo tomar el lector este diagrama como un avance de lo que queda por venir en el resto del capítulo.

4.3 Desarrollo de Software Dirigido por Modelos

En [248] se define el Desarrollo Dirigido por Modelos como la construcción de un modelo de un sistema que más tarde se podrá transformar en algo real. El *Desarrollo de Software Dirigido por Modelos*, por tanto, implica la construcción de un sistema software a partir de modelos.

Actualmente, el DSDM se puede considerar como un nuevo paradigma de desarrollo de software que resulta prometedor para tratar con la complejidad de las plataformas [353], y en el que los modelos, el metamodelado [21] y las transformaciones entre modelos [357] son piezas elementales. El DSDM se basa en una serie de principios fundamentales: (1) Los modelos son tratados como ciudadanos de primera clase. (2) Todo es un modelo. (3) Un modelo se atiene a otro modelo, que se conoce como metamodelo. (4) Una transformación de modelos toma modelos como entrada y produce modelos como salida. (5) Una transformación de modelos también es un modelo.

Dentro del desarrollo de software dirigido por modelos hay varias aproximaciones: SF [151], MDA [282], MIC [196], y otros. En la tabla 4.1 se muestra un resumen de estas aproximaciones, el organismo que está detrás de ellas y algunas de las herramientas disponibles para hacer un desarrollo dirigido por modelos basado en el enfoque concreto.

Software Factories (FS)	Model-Driven Architecture (MDA)	Model-Integrated Computing (MIC)	Otras propuestas
Microsoft	OMG	MIC	Otros
Microsoft Visual Studio DSL Tools	Eclipse EMF GMF	GME	Otras herramientas

Tabla 4.1: Principales aproximaciones al DSDM

Por su relevancia destacan SF y MDA. Aunque en un principio estas dos aproximaciones pudieran parecer incompatibles, se pueden complementar [267, 69]. En primer lugar, Factorías Software es una propuesta relacionada con las líneas de productos, de tal forma que propone el uso de herramientas extensibles y configurables para automatizar el desarrollo y el mantenimiento de diferentes familias de un producto software. Esta automatización se consigue gracias a la configuración y composición de diferentes componentes basados en *frameworks*. Entre las diferentes actividades y técnicas que comprenden las SF se encuentra el desarrollo de diferentes lenguajes de modelado y herramientas concretas para dominios específicos.

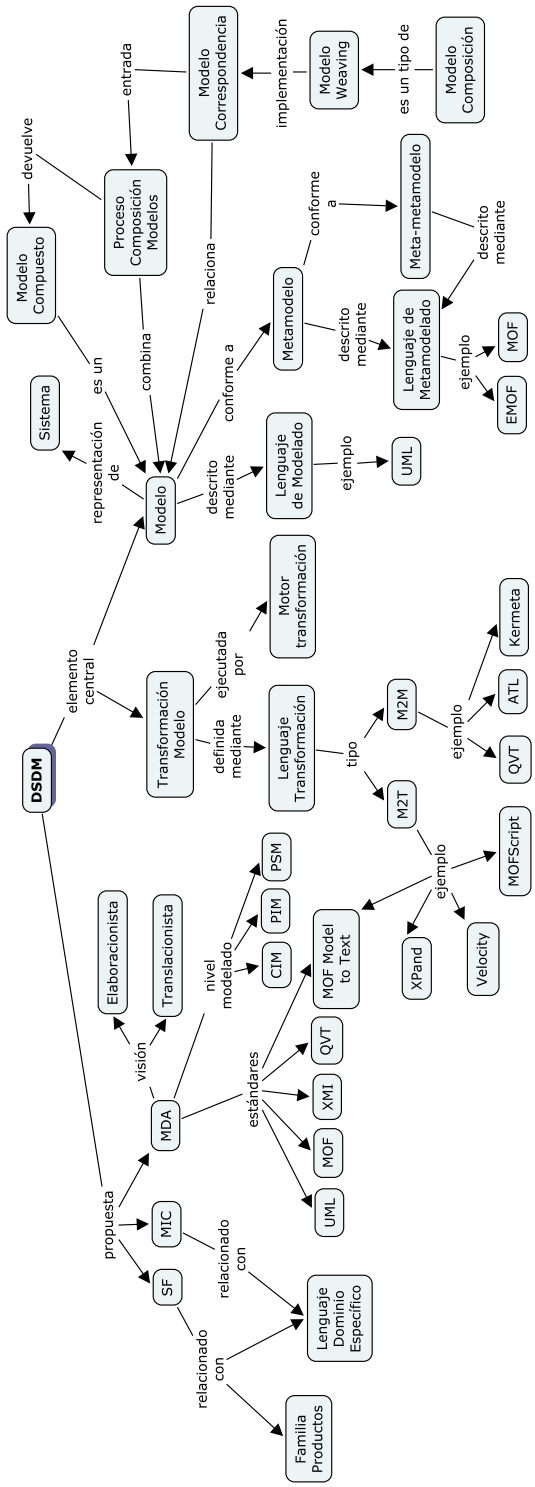


Figura 4.5: Mapa conceptual DSDM

Por otra parte, la propuesta de la OMG se basa en el uso de una serie de estándares emergentes para definir un conjunto de modelos, notaciones y reglas de transformación (UML [284], Meta Object Framework (MOF) [286], QVT [285], ...). De esta propuesta se hablará con más detalle en el apartado 4.5.

Finalmente, la propuesta MIC de la Universidad de Vanderbilt se enfrenta al desarrollo de sistemas integrados de software proporcionando entornos de modelado de dominio específico ricos, que incluyen herramientas de síntesis de programas basadas en modelos y en análisis de modelos. En esta propuesta se utilizan modelos de múltiples vistas para capturar la información relevante para el sistema, se representan las dependencias y las restricciones entre las diferentes vistas de modelado, y automáticamente se sintetizan los diferentes tipos de artefactos software.

4.4 Lenguajes específicos de dominio y modelos

Una de las decisiones que tiene que tomar un ingeniero que se enfrente a un desarrollo dirigido por modelos es si va a utilizar un lenguaje de propósito general, como UML, o bien un Lenguaje Específico de Dominio (DSL). Se puede decir que los DSL's permiten elevar el nivel de abstracción de los programas utilizando conceptos del dominio del problema [382].

Para describir lenguajes y, por tanto, DSL's, se pueden utilizar formalismos de dos espacios técnicos distintos, el de las gramáticas (o *grammarware*) y el de los modelos (o *modelware*). Un espacio técnico, según [228], es un contexto de trabajo, con un cuerpo de conocimiento, herramientas y habilidades requeridas. En el espacio técnico de las gramáticas, se puede usar BNF para describir la gramática del lenguaje, en cuyo caso, tenemos información de la sintaxis concreta, ya que se incluyen palabras clave del lenguaje a la hora de definir la gramática. Mientras que en el espacio técnico de los modelos, el DSL [229] se define como un conjunto de modelos coordinados, de forma que se definen modelos para la sintaxis abstracta, la concreta y la semántica operacional, y una serie de transformaciones de modelos que relacionan la sintaxis abstracta con la concreta y con la semántica operacional.

Uno de los esfuerzos más importantes que están realizado los investigadores en la Ingeniería Dirigida por Modelos es la definición de puentes para integrar los distintos espacios técnicos. Así, por ejemplo, algunos trabajos para acercar el mundo de los modelos y el de las gramáticas son [408] y [409]. En [408] se propone un puente genérico para acercar estos dos espacios tecnológicos, y en [409] también se propone otro puente entre estos dos espacios utilizando la especificación *Human-Usable Textual Notation (HUTN)* [283] de la OMG, un estándar para intercambiar modelos MOF, cuyo objeto es que sean más legibles para los humanos que XMI.

Por último, si el ingeniero ha escogido modelar su problema mediante un DSL y lo va a especificar en el espacio técnico de los modelos, tendrá que decidir si define su lenguaje desde cero utilizando MOF, o va a usar los mecanismos de extensión de UML y

definir un perfil. Ante esta disyuntiva, en la Ingeniería Dirigida por Modelos también se proponen puentes para acercar los perfiles UML y metamodelos MOF [3].

Como conclusión de esta sección, se puede decir que la Ingeniería Dirigida por Modelos es un enfoque integrador que abarca a varios espacios técnicos [228].

4.5 Model Driven Architecture

El principal objetivo de MDA es desacoplar el modo en que se definen las aplicaciones de la tecnología en la que se ejecutan las mismas. Este objetivo se puede alcanzar si se separan las partes dependientes de la plataforma de aquellas que son independientes de la misma, lo que se consigue gracias a los cuatro principios que subyacen bajo la propuesta de la Object Management Group (OMG) [47]:

1. Los modelos expresados en una notación bien definida son la piedra angular para entender el sistema para soluciones de escala empresarial.

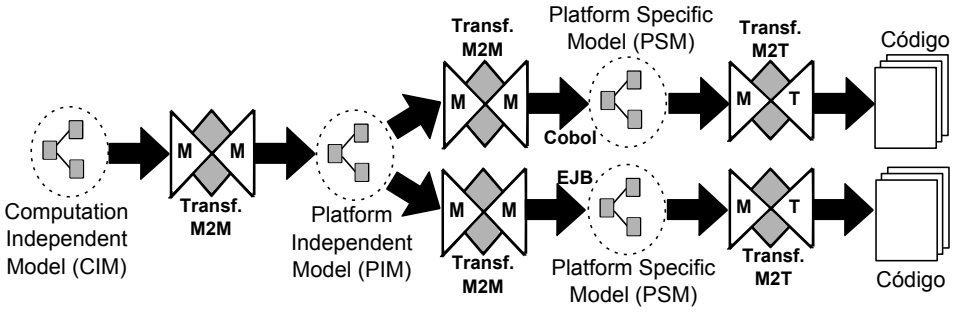
2. La construcción de sistemas se puede organizar alrededor de un conjunto de modelos imponiendo una serie de transformaciones entre modelos, que se organizan en un *framework* con una arquitectura en capas y una serie de transformaciones.

3. Una base formal para describir modelos está en un conjunto de metamodelos que facilitan la integración con significado y la transformación entre modelos, y que constituyen la base para la automatización a partir de herramientas.

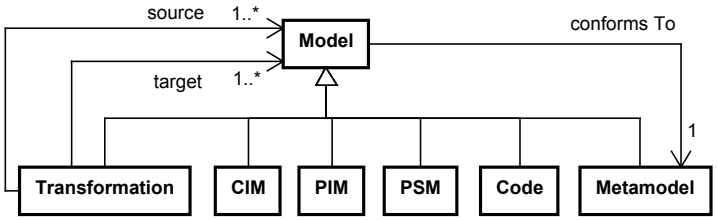
4. La aceptación y la adopción amplia de esta aproximación basada en modelos requiere estándares industriales para proporcionar apertura a los consumidores, y fomentar la competitividad entre los vendedores.

La arquitectura mencionada en el segundo principio está compuesta por diferentes niveles (o capas) de modelos: Modelo Independiente de Computación, Modelo Independiente de Plataforma y Modelo Específico de Plataforma. Según la definición de la OMG [282], un modelo independiente de computación (o CIM, utilizando su abreviatura inglesa) es una vista de un sistema desde un punto de vista independiente de la computación. El CIM es lo que normalmente se conoce como modelo del dominio. Un modelo independiente de plataforma (o PIM, siguiendo nomenclatura inglesa) es una vista de un sistema desde un punto de vista independiente de la plataforma. Es decir, el PIM es un modelo que puede utilizarse con plataformas de diferentes tipos. Finalmente, un modelo específico de plataforma (o PSM) es una vista de un sistema desde un punto de vista de una plataforma específica. En el PSM se combina la especificación de un PIM con detalles de una plataforma concreta. En la figura 4.6(a) se muestran estos modelos y las relaciones entre ellos. La notación usada en la figura, se ha adoptado de la usada por Gronback [156] en su libro. Note que para obtener un modelo PIM a partir de un modelo CIM habrá que definir una serie de transformaciones de modelo a modelo. De

forma equivalente, se puede obtener el modelo PSM a partir de un modelo independiente de plataforma definiendo otra serie de transformaciones modelo-modelo. Así, por ejemplo, la figura 4.6(a) refleja el hecho de que a partir del mismo PIM, y con dos conjuntos de transformaciones diferentes, se pueden obtener modelos específicos para COBOL o para Enterprise Java Beans (EJB). Por último, a partir de estos modelos con detalles concretos de las plataformas, y definiendo una serie de transformaciones modelo a texto, se obtiene el código generado.



(a) Representación visual



(b) Metamodelo

Figura 4.6: Elementos de la propuesta MDA

Como resumen la figura 4.6(b) muestra un metamodelo que describe los principales elementos que intervienen en MDA. Observando estas dos figuras se puede comprobar que MDA ha hecho que las transformaciones se conviertan en elementos clave del proceso de desarrollo, ya que son el mecanismo utilizado para obtener un modelo a partir de otro. Además de esta separación de conceptos en los distintos niveles de modelos, la OMG recomienda trabajar con una serie de estándares. En la figura 4.2 se mostraban los estándares propuestos como lenguajes de modelado. En este caso la OMG recomienda UML [284] como lenguaje de modelado, MOF [286] como lenguaje de metamodelado, y XML Metadata Interchange (XMI) [287] para serializar los modelos y metamodelos y poderlos compartir entre distintas herramientas. Finalmente, el creciente interés en las transformaciones de modelos ha hecho que la OMG proponga Queries Views Transformations (QVT) [285] como lenguaje de transformación de modelo a modelo y OMG Model to Text [288] para las transformaciones de modelo a texto. En la figura 4.7, se muestra un esquema de los estándares de lenguajes de transformaciones propuestos por la OMG.

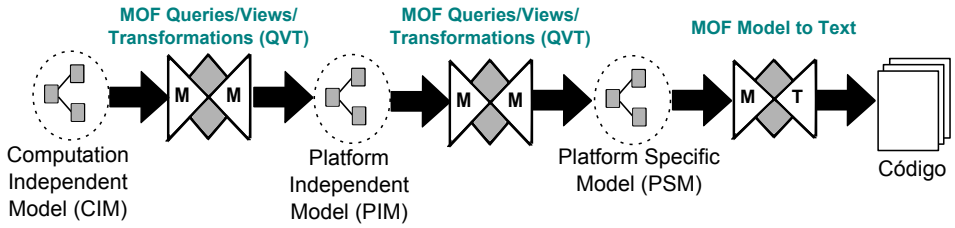
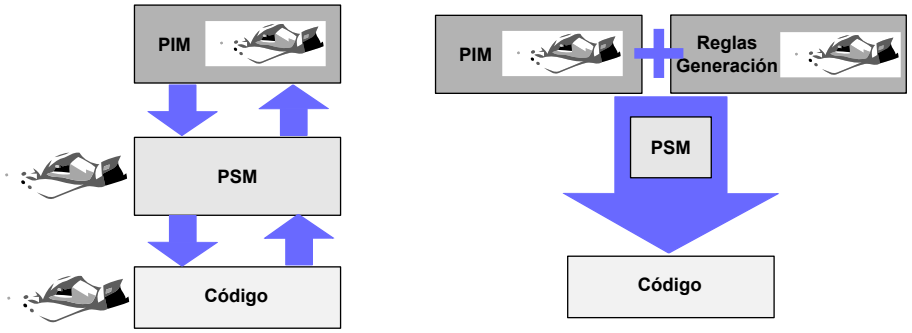


Figura 4.7: Estándares OMG relacionados con las transformaciones

Aunque la OMG ha definido MDA, existen diferentes interpretaciones de la misma (figura 4.8): el enfoque elaboracionista, que es la que más se parece a la descripción de MDA que aparece en el documento de la OMG, y que se puede resumir en el libro publicado por Anneke Kepple et al. [215]; y el enfoque translacionista, respaldado por Stephen Mellor en [247].

En el *enfoque elaboracionista* la definición de la aplicación se construye gradualmente conforme se va progresando de PIM a PSM y luego a código. Una vez que se ha creado el PIM, la herramienta genera un primer esqueleto del PSM que el desarrollador puede "elaborar" añadiendo más información o más detalles. De igual forma, la herramienta genera el código final del PSM, y esto también puede ser elaborado. El que se requiera elaboración en ambos niveles (PSM y código) dependerá de la herramienta y de las circunstancias. Como resultado de la elaboración puede ocurrir que los modelos de más bajo nivel dejen de estar "sincronizados" con los de más alto nivel. Por esto, las herramientas deberían tener la capacidad de generar modelos de más alto nivel a partir de los de más bajo nivel. Claramente, en este enfoque, los artefactos generados (PSM y código) deben ser entendibles por el desarrollador, si no, la modificación (elaboración) no sería posible. Este enfoque representa la corriente dominante en MDA.

Por contra, en el *enfoque translacionista* el PIM se traduce directamente al PSM y al código. Las reglas de generación se han de implementar en un traductor y solamente se pueden formular para un dominio específico. En este enfoque, el PIM se traduce directamente en el código final del sistema mediante generación de código. La generación (o traducción del PIM) en el código final es realizada por un generador de código, muchas veces denominado "compilador de modelos". El proceso es guiado por reglas de generación que describen cómo se representan los elementos del PIM en el código final. En este caso, el PSM es una etapa intermedia en la generación y es interno al generador de código. Generalmente no es visible ni editable por el desarrollador. Una característica de este enfoque es que el PSM y el código no son elaborados o retocados a mano.



(a) Elaboracionista. (b) Translacionista

Figura 4.8: Las dos interpretaciones de MDA

4.6 Técnicas de transformación de modelos

La tecnología de transformaciones ha evolucionado mucho desde sus comienzos. Así, en [52] se propone una clasificación de los *frameworks* de transformaciones en tres generaciones, dependiendo del grado de evolución de la tecnología utilizada.

Como primera generación se pueden considerar las utilidades del estilo de `awk` [91], que aparecen con el sistema operativo UNIX. En esta primera generación, un archivo de texto contenía un *script* de transformación, en el que se expresaba cómo se podía obtener un archivo objeto a partir de un archivo fuente. En la segunda generación se pasa a trabajar con árboles en lugar de con archivos, de tal forma que mediante un *script* se indica cómo transformar un árbol de entrada en un árbol de salida. En esta segunda generación se trabaja con sistemas de transformación tales como `XSLT` [396] o XQuery [400]. Se tiene un lenguaje (como XPath [399]) que se utiliza para formular expresiones de navegación en el árbol. Navegar en el árbol origen producirá trozos del árbol destino. Finalmente, en la actualidad se puede considerar que estamos en la tercera generación, en la que se ha pasado de trabajar con archivos o árboles, a hacerlo con grafos, pero restringidos por un metamodelo común.

Como las estrategias para realizar transformaciones difieren dependiendo de si los artefactos a generar son textuales o modelos, en el resto de la sección se detallarán por separado las técnicas existentes para realizar transformaciones de modelo a texto (sección 4.6.1) y de modelo a modelo (sección 4.6.2).

4.6.1 Técnicas de transformación modelo-texto

El objetivo de las *transformaciones de modelo a texto (o transformaciones `M2T`)* es la generación de artefactos textuales a partir de la información subyacente en los modelos. Estos artefactos textuales incluyen cosas tales como código fuente, archivos `XML`, archivos `HTML`, documentos. En el marco de `MDA` las transformaciones de modelo a texto se utilizan para refinar los modelos específicos de plataforma (`PSM`'s) y obtener finalmente

el código fuente (Java, C++) y los archivos de configuración y recursos asociados (XML, etc.).

Un motor de transformaciones de modelo a texto toma como entradas uno o varios documentos con la descripción de las transformaciones, uno o varios modelos a los que se le van a aplicar las transformaciones, y los metamodelos que restringen a los modelos; y produce como resultado uno o varios artefactos textuales resultado de las transformaciones. En la figura 4.9 se muestra un motor de transformación genérico que toma como entrada un metamodelo que es una versión simplificada de un diagrama de clases UML y en el que solamente aparecen cinco metaclases Classifier, PrimitiveDataType, Class, Attribute y Package. El modelo se ha representado con un diagrama de objetos para dejar explícita la relación "instancia de" con el metamodelo. El modelo contiene un paquete llamado App, con dos clases Customer y Address. Customer tiene dos atributos name y addr, para recoger el nombre y la dirección del cliente, respectivamente; mientras que Address solamente tiene el atributo addln. En este caso, la transformación modelo a texto deberá hacer algo como lo que se describe a continuación: "Por cada instancia de la metaclase Class se genera un archivo .java cuyo nombre sea el valor del atributo name de Class. En el archivo .java, lo primero que debe aparecer es la palabra clave package seguida del valor del atributo name de la metaclase Package, siempre y cuando exista una relación entre la clase y el paquete, ...". El resultado de la transformación para Customer se representa como salida del motor de transformación.

Tradicionalmente se han seguido dos tipos de técnicas [78] para realizar las transformaciones modelo-texto, las basadas en visitante, y las basadas en plantillas. A continuación se describe con un poco de más detalle en qué consiste cada tipo de técnica:

Técnicas M2T Basadas en Visitante. Son el tipo de aproximación más básico para la generación de código y consiste en proporcionar algún mecanismo de visitante para ir recorriendo la representación interna de un modelo, e ir escribiendo así el texto generado en algún archivo.

Dentro de este grupo nos podemos encontrar tanto programas escritos en cualquier lenguaje de programación que permita manipular cadenas y acceder a la representación de interna del modelo, como propuestas basadas en XSLT [396]. En este último caso, mediante una hoja de estilo se puede transformar un documento XML en código fuente.

Una propuesta que utiliza esta estrategia es Jamda [42], un *framework* para construir generadores de código dentro del marco de MDA. Jamda toma una serie de modelos UML como entrada, les añade algunas clases para poder hacer la generación, y finalmente genera código ejecutable. Ejemplos de generadores de código basados en XSLT son [346] y [40].

Técnicas M2T Basadas en Plantillas. Este tipo de estrategia normalmente hace uso de *plantillas*. Una plantilla puede definirse como una secuencia de texto entremezclada con comandos que extraen información de un modelo. Este metacódigo no

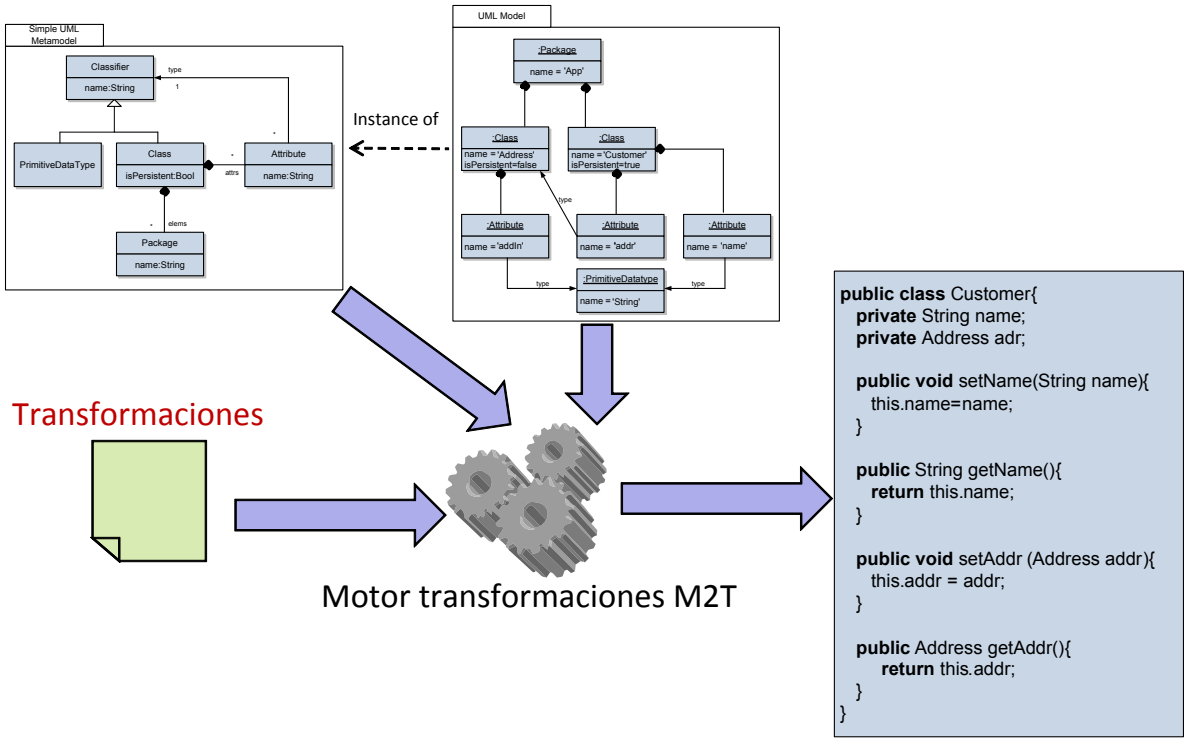


Figura 4.9: Ejemplo de transformación modelo texto basado en [78]

se limita a la extracción de información, sino que a menudo sirve como flujo de control en la plantilla, indicando si un bloque de texto se procesa o no, o cuántas veces se ha de procesar.

La estructura de una plantilla se parece bastante al texto que se va a generar, aunque es independiente del lenguaje objetivo, simplificando bastante la generación de cualquier artefacto textual. La mayoría de herramientas que siguen la filosofía propuesta en `MDA` se han adherido a esta aproximación.

Dentro de las estrategias basadas en plantillas también se pueden distinguir dos tipos: las que se basan en plantillas más una técnica de filtrado, y las que se basan en plantillas y metamodelos. En las primeras el código se genera aplicando las plantillas a especificaciones textuales de modelos (a menudo en `XML` o `XMI`), y lo que se hace es filtrar alguna parte de la especificación. En las segundas, sin embargo, en lugar de aplicar las plantillas directamente al modelo, primero se instancia un metamodelo a partir de la especificación del mismo. En este caso, las plantillas se especifican en términos del metamodelo.

Una tecnología relacionada con la generación de código, aunque no está basada en modelos es el procesamiento de marcos (en inglés, *frame processing*). Los marcos pueden verse como programas o funciones que generan código como resultado de su evaluación,

y extienden a las plantillas con mecanismos de estructuración y adaptación más avanzados. Ejemplos de esta tecnología son los marcos de Bassett [29] y XFramer [415, 105].

4.6.2 Técnicas de transformación modelo-modelo

Las *transformaciones de modelo a modelo (o transformación M2M)* se pueden utilizar para: transformar modelos en el marco de la propuesta MDA, como por ejemplo, transformar modelos PIM en modelos PSM; refinar modelos; hacer ingeniería inversa; generar nuevas vistas de un sistema; aplicar patrones de diseño; refactorizar modelos; derivar productos en las líneas de producto o, en definitiva, para cualquier actividad que se pueda automatizar dentro de la ingeniería de modelos.

Un motor de transformaciones de modelo a modelo toma como entradas uno o varios documentos con la descripción de las transformaciones, uno o varios modelos a los que se le van a aplicar las transformaciones, y los metamodelos que restringen a los modelos, y produce uno o varios modelos resultado de las transformaciones. En la figura 4.10 se muestra un motor de transformación genérico que toma como entradas un metamodelo simple de UML (el mismo que se utilizó en la figura 4.9), un modelo UML conforme a este metamodelo (que ya ha sido descrito en la sección 4.6.1 cuando se explicaba la figura 4.9), y un metamodelo simple de un esquema relacional de bases de datos. El modelo resultado de las transformaciones tendrá que ser conforme a este metamodelo.

En el metamodelo Simple RDBMS hay definidas cuatro metaclases: Schema, Table, Column, y FKey. Estas metaclases representan, respectivamente, a un esquema de la base de datos, una tabla, un campo y una clave ajena. En el metamodelo, además de las metaclases, se representan las relaciones entre ellas. Así, se puede observar que un Schema está compuesto por un conjunto de Table. Una tabla contiene un conjunto de campos y claves ajenas. En las transformaciones que se dan como entrada al motor de transformación se tendrá que expresar algo como lo siguiente: "Por cada instancia de la metaclase Package que pertenece al metamodelo simple de UML, generar una instancia de la metaclase Schema del metamodelo Simple RDBMS. La propiedad name de la instancia de Schema tendrá el mismo valor que la propiedad name de la instancia de Package. Por cada instancia de la metaclase Class cuya propiedad isPersistent tenga un valor cierto, generar una instancia de la metaclase Table..."

En la actualidad, nos podemos encontrar con seis tipos de técnicas distintas para realizar transformaciones de modelo a modelo [78]: aproximaciones de manipulación directa, aproximaciones de transformación genéricas, aproximaciones basadas en transformaciones de grafos, aproximaciones operacionales, aproximaciones basadas en plantillas, aproximaciones relacionales, y aproximaciones híbridas. A continuación se muestran las características principales de cada una de ellas.

Técnicas de manipulación directa de modelos. Las herramientas o *frameworks* que utilizan esta técnica se basan en proporcionar una forma de acceder a la representación interna propia que tienen de los modelos. Normalmente, este acceso se

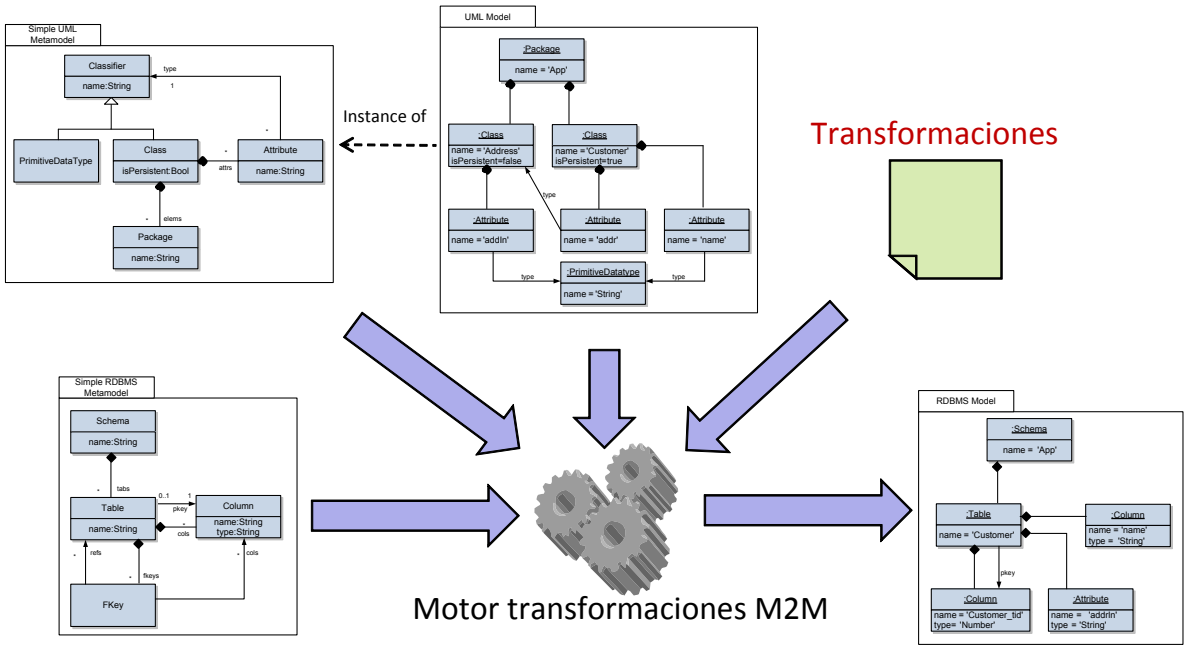


Figura 4.10: Ejemplo de transformación modelo modelo basado en [78]

consigue gracias a una Interfaz de Programación de Aplicaciones (`API`) que permite realizar manipulaciones de los modelos descritos en esa representación interna. El lenguaje utilizado para expresar las transformaciones suele ser un lenguaje de de propósito general como Java, Visual Basic, C++ o C#, y éstas se implementan desde cero utilizando el lenguaje de programación y la infraestructura proporcionada por la propia `API`.

Técnicas operacionales. La idea que subyace en este tipo de técnicas es muy parecida a la de las técnicas de manipulación directa, pero en este caso lo que se suele hacer es extender el lenguaje con el que se describe el metamodelo con facilidades para expresar los cálculos.

Técnicas de transformación genéricas. En esta categoría se encuentran aquellas herramientas que en un principio no se pensaron para realizar transformaciones de modelos pero que, sin embargo, se pueden utilizar para realizar estas transformaciones. Ejemplos de este tipo de herramientas son los comandos Unix `awk` o `sed` [91]; el lenguaje PERL [355]; transformaciones implementadas utilizando `XSLT` [396] para documentos `XML`; o la aplicación de la metaprogramación a las transformaciones de modelos.

Las transformaciones implementadas en `XSLT` se pueden utilizar con los modelos en formato `XMI` [287], ya que hay muchas herramientas `UML` que pueden importar y exportar modelos en este formato. Así, se pueden utilizar las herramientas `XML` ya

existentes con los modelos expresados en XMI. Desde esta perspectiva, los modelos pueden verse como árboles XML, y con XSLT transformamos estos árboles XML en otros árboles XML.

Técnicas basadas en transformaciones de grafos. Los modelos desde la perspectiva de las herramientas de transformación de grafos son grafos etiquetados, atribuidos y con tipos. Este tipo de herramientas suelen ser declarativas y tienen su base teórica en los trabajos de transformación de grafos. Normalmente trabajan con reglas que tienen parte derecha, o *Right Hand Side (RHS)* y parte izquierda, o *Left Hand Side (LHS)*, y que definen un patrón sobre el grafo. Cuando el patrón definido por la parte izquierda concuerda con alguna parte del modelo a transformar, esta parte del modelo se sustituye por la parte derecha de la regla. Normalmente, en la parte izquierda se suele incluir alguna condición además del propio patrón.

Técnicas guiadas por la estructura. Este tipo de técnicas se ejecutan en dos fases. En una primera fase se crea la estructura jerárquica del modelo destino, mientras que en la segunda fase se asignan los atributos y las referencias en el modelo destino. Normalmente, las herramientas que utilizan este tipo de técnicas tienen un enfoque pragmático desarrollado en el contexto de EJB y la generación de esquemas de bases de datos a partir de modelos UML.

Técnicas basadas en plantillas. La idea es parecida a la mostrada en el apartado 4.6.1 cuando se explicaron las transformaciones modelo a texto basadas en plantillas. En este caso, la plantilla en lugar de ser una secuencia de texto con huecos, es un modelo con "huecos". Los huecos contienen metacódigo que sirve para calcular la parte variable del modelo objetivo. Normalmente, el modelo con los "huecos" o plantilla tiene la forma del modelo resultado de la transformación y el metacódigo suelen ser anotaciones.

Técnicas relacionales. Este tipo de técnicas presentan enfoques declarativos basados en el concepto matemático de relación. Este tipo de aproximaciones puede verse como una forma de resolver restricciones. La idea es expresar las relaciones entre los elementos del modelo origen y el modelo destino utilizando restricciones.

Una forma de implementar este tipo de técnicas es mediante programación lógica, de tal forma que las relaciones se describen mediante predicados.

Técnicas híbridas. En este grupo están las aproximaciones que combinan diferentes técnicas de las presentadas en las categorías anteriores. Las distintas técnicas se pueden combinar como componentes separados, como ocurre en QVT [285], que tiene tres componentes separados, Relations, Operational Mappings y Core; o bien, ser mezclados a nivel de reglas individuales, como ocurre con ATL [94], que proporciona reglas de transformación declarativas, híbridas o imperativas.

4.7 Técnicas de composición de modelos

Un algoritmo de composición de modelos define la semántica de la relación de composición entre modelos y especifica cómo se deben manipular los modelos de entrada con el objeto de componerlos. En la bibliografía hay un amplio abanico de algoritmos de composición que van desde algoritmos manuales hasta completamente automatizados. Pero, en general, se puede decir que la composición de modelos es un proceso que está formado por dos actividades claves [121, 344]:

- Matching (o emparejamiento). En esta fase se establecen las correspondencias entre los elementos de los modelos fuentes que se han identificado como similares. Es decir, se determinarían qué elementos se identifican como equivalentes en el proceso de emparejamiento. El emparejamiento se puede conseguir de forma manual, o bien, utilizando alguna heurística como, por ejemplo, el emparejamiento por nombre, con lo que se definirían correspondencias entre aquellos elementos que se llaman igual.

- Composición. En esta fase se utilizan las correspondencias definidas en la fase anterior para producir una versión unificada del modelo.

Como ya se ha comentado anteriormente, la composición de modelos es una operación que se utiliza en diferentes áreas de conocimiento, lo que ha provocado que no haya un vocabulario común y que sea difícil comparar las propuestas provenientes de las distintas áreas. Una primera aproximación para definir una terminología común es la que propone Bézivin en [51], mientras que en [199] se define un marco para comparar la gran variedad de propuestas que hay en la bibliografía. Este marco se centra en tres preguntas que se deben responder en cualquier algoritmo de composición: ¿Qué elementos hay que componer? ¿Dónde hay que colocar el resultado de la composición en el modelo o modelos resultado? y ¿Cómo se realiza la composición? La primera pregunta implica decidir cuáles de los elementos de los modelos se compondrán y cuáles no. La segunda requiere seleccionar una localización en el modelo o modelos resultado en la que se realizará la modificación o la inserción. Finalmente, con la tercera se especifica cómo combinar e integrar los conceptos seleccionados en el modelo o los modelos resultados.

Además, de estas tres cuestiones generales, en el marco se especifica con más detalle el proceso de composición, dividiéndolo en varias fases y definiéndolo de forma iterativa, tal y como se muestra en la figura 4.11. Es interesante este refinamiento porque la fase de traducción es necesaria cuando se trabaja con diferentes lenguajes de modelado.

El proceso de referencia es iterativo, y en cada iteración se acometen las fases representadas en la figura 4.11. En primer lugar se seleccionan una serie de elementos de los modelos de entrada (en esta fase se responde a la pregunta ¿qué?). Luego se selecciona en qué lugar del modelo o modelos destino se colocaran los elementos resultado de la composición. Finalmente, a la pregunta cómo se responde en tres etapas, que van desde combinar los elementos de los modelos, traducir el resultado de la combinación para

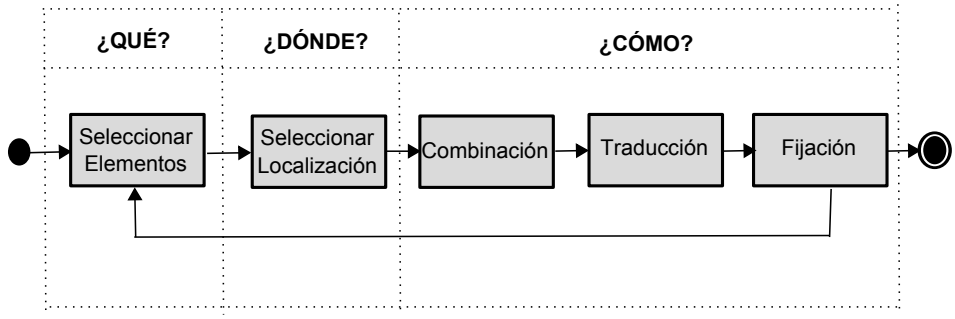


Figura 4.11: Proceso de referencia para la composición de modelos de Jeanneret

que sea compatible con el metamodelo destino (en el caso en que se trabaje con diferentes metamodelos origen y destino) y, finalmente, fijar el nuevo elemento al modelo resultado, en el sentido de que a veces habrá que conectarlo con el resto de elementos del modelo resultado ya calculados.

Por último, hay que señalar que de las once propuestas analizadas por Jeanneret usando su marco de comparación, solamente hay dos que tienen en cuenta la fase de traducción: Atlas Model Weaver (AMW) [85, 93] y Epsilon Merging Language (EML) [96].

4.8 Estándares y entorno tecnológico

En los últimos años se ha producido una explosión de herramientas, lenguajes y estándares para enfrentarse al Desarrollo de Software Dirigido por Modelos, y especialmente, aquellos relacionados con la propuesta MDA. Este apartado no pretende ser exhaustivo, ni dar una comparativa de todas las propuestas existentes en la bibliografía, puesto que ya hay publicadas varias, sino que intenta dar al lector una visión general de aquellas herramientas más maduras o que han tenido más difusión. En los siguientes apartados de esta sección se enumeran las herramientas y estándares disponibles relacionadas con los lenguajes de descripción de metamodelos, los lenguajes de transformación de modelo a texto, las herramientas de transformación de texto a modelo, los lenguajes de transformación de modelo a modelo y las herramientas de composición de modelos.

4.8.1 Lenguajes de descripción de metamodelos

En el DSDM los metalenguajes tales como MOF [286] se utilizan para especificar metamodelos. La persistencia de los mismos se consigue vía serialización en XML (XMI [287] es el estándar propuesto por la OMG como formato de intercambio de modelos), o a través de un almacenamiento directo en una base de datos (como por ejemplo, Metadata Repository (MDR) [273], que es un repositorio para metadatos descritos en MOF). Además del estándar definido por la OMG, en la industria ha surgido un estándar de facto, más

ligero, para definir metamodelos como resultado del proyecto Eclipse Modelling Framework (EMF) [311, 49]. EMF es, por una parte, un *framework* y, por otra parte, una utilidad para definir modelos en uno de los siguientes formatos: como interfaces Java, como diagramas UML o como XML Schemas. Ecore es el metamodelo que utiliza EMF para representar modelos. Ecore está definido de forma que también es un modelo Ecore, así que se puede decir está al mismo nivel que MOF dentro de los niveles de modelado definidos por la OMG (ver figura 4.2).

Lenguaje	Ref	Herramienta
MOF	[286]	Estándar
EMF Ecore	[49]	EMF [311]
KM3	[202]	AMMA[157]
MetaGME	[104]	GME [195]
SMD	[7]	Coral [74]
Kermeta	[264]	Kermeta [160]
Microsoft DSL Tools Metamodel	[70]	DSL Tools [253]

Tabla 4.2: Lenguajes de descripción de metamodelos

En la tabla 4.2 se muestra un resumen de los principales lenguajes de descripción de metamodelos. En la primera columna aparece el acrónimo del lenguaje, en la segunda una referencia con más detalles del mismo, y en la tercera, el nombre de la herramienta que utiliza el lenguaje así como una referencia desde la que se puede acceder a la herramienta. Si observamos la tabla por filas, en primer lugar aparece MOF, el estándar de la OMG. Luego, está el estándar de facto *EMF Ecore*, que se puede considerar una implementación reducida de MOF. Aunque estos dos lenguajes no tienen la misma expresividad, se puede definir una equivalencia entre los mismos [255, 138, 92].

KM3 viene de *Kernel MetaMetaModel* y proporciona una sintaxis concreta simple con formato textual para facilitar la codificación de los metamodelos. La notación KM3 se parece a la notación Java. Sus ficheros, que tienen extensión .km3, se pueden transformar en XMI. MetaGME es el lenguaje de metamodelado que da soporte a GME y está basado en diagramas de clases con estereotipos y restricciones OCL. También se puede definir una equivalencias entre modelos MetaGME y Ecore [37]. SMD son las siglas de *Simple Metamodel Description language*, el lenguaje de metamodelado que acompaña al *framework* Coral. Kermeta, aunque no es estrictamente un lenguaje de metamodelado puesto que extiende a MOF con operaciones para definir la semántica operacional, permite también definir metamodelos. Finalmente, las DSL Tools de Microsoft también proporcionan su propio lenguaje de metamodelado, que se puede traducir a EMF [38].

4.8.2 Lenguajes de transformación de modelo a texto

El estándar propuesto por la OMG salió a finales de Noviembre de 2006, y se conoce como MOF Model to Text Transformation Language [288]. Algunos de los lenguajes de transformación de modelo a texto que se van a ver en este apartado se enviaron a la

OMG cuando la organización realizó en 2004 su petición de propuestas para estandarizar el lenguaje de transformación de modelo a texto. En la tabla 4.3 se muestra un resumen de los lenguajes de transformación de modelo a texto más populares. La primera columna de la tabla representa el nombre o las siglas por las que que conoce al lenguaje, la segunda columna contiene una referencia donde se dan más detalles del lenguaje, y en la última columna aparece la herramienta o el proyecto dentro del marco Eclipse con el que se distribuye el lenguaje. En este punto cabe destacar que, a diferencia de lo que ocurría con los lenguajes de descripción de metamodelos, en los que EMF Ecore se ha establecido como estándar de facto, no se puede afirmar que alguno de los lenguajes de transformación de modelo a texto definidos en el marco Eclipse se haya convertido en estándar de facto.

Lenguaje	Ref	Herramienta
MOF Model to Text	[288]	Estándar
JET	[305]	Proyecto Eclipse EMF [383]
MOFScript	[281]	Proyecto Eclipse GMT [312]
VTL	[379]	AndroMDA [19]
xPand	[100]	openArchitectureWare [292]

Tabla 4.3: Lenguajes de transformación de modelo a texto

Java Emitter Templates (JET) forma parte del proyecto EMF [311] y es una de las herramientas de generación de código incluidas en este proyecto. JET tiene una sintaxis similar a JSP que hace que sea fácil el escribir las plantillas que expresan el código que se desea generar. Actualmente JET está bajo el proyecto *Model to Text (M2T)* de Eclipse [310], que tiene como objetivo proporcionar implementaciones de los estándares industriales y de facto de motores de transformación modelo a texto, proporcionar herramientas para estos lenguajes y una infraestructura común para los mismos. Dentro de este proyecto también está xPand, el lenguaje propuesto en el *framework* de *openArchitectureWare*. El lenguaje define algunos conceptos de orientación a aspectos, tales como puntos de corte (*pointcut*) o *advice*.

MOFScript es la respuesta enviada por el proyecto *Modelware* de *Eclipse GMT* a las necesidades de estandarización del lenguaje de transformación de modelo a texto lanzada por la OMG. MOFScript utiliza una herramienta específica basada en metamodelos. El lenguaje MOFScript incluye la generación tanto a archivos como por la salida estándar, asignación de variables, lógica de control, funciones de biblioteca y reglas para transformación.

Velocity Template Language (VTL) es un lenguaje muy popular debido a la simplicidad de su sintaxis y a su potencia, ya que proporciona mecanismos de extensibilidad avanzados. Velocity es un motor de generación basado en plantillas y de código abierto que es usado por varios generadores de código, entre los que se encuentra AndroMDA.

4.8.3 Herramientas de transformación de texto a modelo

Con este tipo de herramientas, Eclipse se ha adelantado a la OMG, y ha lanzado una solicitud de peticiones para un proyecto que se llama *Textual Modelling Framework (TMF)*. Si consideramos que un metamodelo es la sintaxis abstracta de un lenguaje, podemos decir que EMF se encarga de tratar con la misma. Pero hay poco soporte para la sintaxis concreta, es decir, las notaciones concretas con las que se representa un modelo. Por ejemplo, en el metamodelo de UML habrá un elemento que represente una clase, y una notación concreta para la clase será un recuadro (si hablamos de lenguajes visuales). El proyecto Textual Modelling Framework (TMF) se encarga de la sintaxis textual, y actualmente, recoge dos aproximaciones, TCS y Xtext, que se muestran en la tabla 4.4. Dentro del entorno Eclipse también ha aparecido EMFText. Una clasificación detallada de las propuestas que trabajan con una sintaxis concreta textual se puede encontrar en [144].

Lenguaje	Ref	Herramienta
TCS	[203]	Proyecto Eclipse GMT [377]
Xtext	[291]	openArchitectureWare [290]
EMFText	[178]	Proyecto Eclipse Reuseware [162]

Tabla 4.4: Herramientas de transformación de texto a modelo

Textual Concrete Syntax (TCS) es una propuesta enviada a la llamada de peticiones realizada por el *Eclipse Textual Modeling Framework (TMF)*. TCS permite la especificación de sintaxis concretas textuales para lenguajes específicos de dominio mediante el enlace de información sintáctica a los metamodelos. TCS permite realizar transformaciones de modelo a texto, así como el análisis de texto para transformarlo en modelos. La idea que subyace en esta propuesta es asociar elementos sintácticos a elementos del dominio del metamodelo, de forma que con esta información se pueda generar una gramática ANTLR. El analizador sintáctico o *parser* derivado de esta gramática es capaz de trabajar con una especificación textual del Domain Specific Language (DSL) y generar su representación como un modelo conforme al metamodelo.

Xtext es la propuesta de openArchitectureWare y, a diferencia de TCS, la estrategia de Xtext es derivar el metamodelo a partir de un archivo de gramática XText. La gramática describe la sintaxis concreta del metamodelo, y posteriormente se transforma a una gramática ANTLR y a un metamodelo Ecore. El analizador sintáctico generado para la gramática ANTLR crea elementos como modelos conformes al metamodelo.

EMFText es una propuesta que se inició como parte del proyecto Reuseware. Es una herramienta que permite, gracias a su mecanismo de importación, modularizar, extender y reutilizar (parcialmente) la especificación de la sintaxis, lo que pone el foco en el refinamiento, la derivación y la evolución.

4.8.4 Lenguajes de transformación de modelo a modelo

El estándar propuesto por la OMG para realizar transformaciones de modelo a modelo es QVT. QVT no es un lenguaje, si no que define tres lenguajes específicos para el dominio de las transformaciones, llamados Relations, Core y Operational Mappings y están organizados en una arquitectura estratificada de lenguajes. QVT/Relations y QVT/Core son lenguajes declarativos definidos a dos niveles de abstracción distintos. QVT también define una serie de transformaciones para pasar de un lenguaje a otro. QVT/Operational Mappings es un lenguaje imperativo que extiende a Relations y a Core. En la figura 4.12[1] , se muestra un esquema de esta arquitectura de lenguajes, mientras que en la tabla 4.5 se pueden ver algunas herramientas que implementan estos lenguajes.

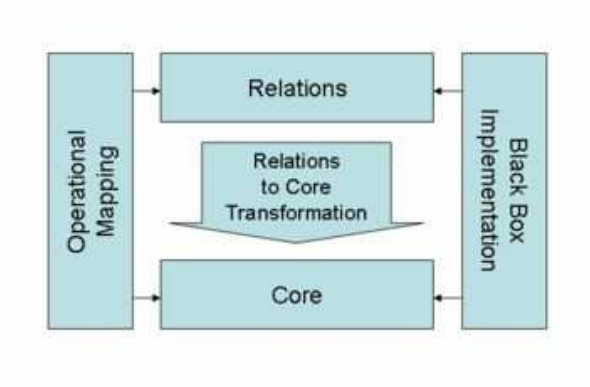


Figura 4.12: Arquitectura en capas de los lenguajes definidos en QVT

Lenguaje		Ref	Herramienta
QVT		[285]	Estándar
	QVT-Relations		Moment [159]
			MediniQVT [191]
	QVT-Core		OptimalJ [65]
	QVT-Operational Mapppings		SmartQVT [101]
ATL		[204]	Proyecto Eclipse Modeling [94]
Kermeta		[264]	Kermeta [160]
SDM		[12]	MOFLON [375]
RubyTL		[345]	AGE [376]

Tabla 4.5: Lenguajes de transformación de modelo a modelo

Actualmente, no hay ninguna herramienta que implemente los tres lenguajes de transformación de la especificación QVT. QVT/Relations se ha implementado en MOMENT, una herramienta que se proporciona como *plug-in* de Eclipse y que está construida sobre Maude [279], un sistema de reescritura de términos que proporciona un

[1] Imagen tomada de http://en.wikipedia.org/wiki/QVT

soporte formal a las operaciones de transformación. Otra implementación de Relations es medini QVT, que también funciona como *plug-in* de Eclipse y dispone de una versión gratuita para uso no comercial. SmartQVT es una implementación de código abierto de QVT/Operational Mappings, mientras que OptimalJ, el producto desarrollado por Compuware, está desarrollando una versión de Core.

Además de estas implementaciones, hay en el mercado otros lenguajes de transformación, algunos de los cuales cumplen parcialmente las especificaciones de QVT. En la tabla 4.5, se muestra más información sobre cada uno de ellos. Al igual que en el resto de subapartados, en la primera columna de la tabla aparece el nombre o las siglas que hacen referencia al lenguaje, en la segunda columna, una referencia bibliográfica, por si el lector quiere obtener más información acerca del lenguaje, y, finalmente, la herramienta en la que se implementa junto con su referencia.

Uno de los lenguajes de transformación de modelo a modelo más populares, quizás porque fue uno de los primeros que venía acompañado por una herramienta, es ATL. ATL viene de *Atlas Transformation Language* y se desarrolló en el INRIA como respuesta a la solicitud de peticiones realizada por la OMG para definir el estándar QVT. Actualmente, se encuentra dentro del proyecto de modelado de Eclipse. Se puede decir que Atlas Transformation Language (ATL) es un lenguaje de transformación híbrido, ya que proporciona constructores imperativos y declarativos. Una vez compiladas las transformaciones, éstas se ejecutan en una máquina virtual (ATL VM).

Kermeta es una extensión a *Essential Meta-Object Facilities (EMOF)* 2.01, una parte de MOF 2.0, para dar soporte a la definición de comportamiento. Como se comentó en la sección 4.6.2, pertenece al grupo de lenguajes de transformación de modelo a modelo operacionales, ya que proporciona un lenguaje de acciones para especificar el cuerpo de las operaciones en los metamodelos. En el caso de Kermeta, este lenguaje es imperativo y orientado a objetos. La plataforma Kermeta se ha distribuido como *plug-in* de Eclipse y se ha desarrollado en el INRIA.

MOFLON es un framework de metamodelado construido sobre algunos componentes de FUJABA [158], y que se basa en reglas de reescritura de grafos. *Story Driven Modelling (SDM)* es el mecanismo definido para realizar las transformaciones de grafos en Fujaba.

RubyTL es un lenguaje de transformación híbrido definido como un lenguaje específico del dominio embebido en el lenguaje de programación Ruby, y diseñado como un lenguaje extensible en el que un mecanismo de plug-ins permite añadir nuevas características al núcleo de características básicas. El entorno de desarrollo creado para RubyTL se llama *Agile Generative Enviroment (AGE)* y consta de un editor con resaltado de sintaxis y asistencia, una herramienta de configuración de plug-ins, un motor de plantillas para las transformaciones modelo-código y un editor de metamodelos.

Finalmente, debido a la gran cantidad de lenguajes de transformación de modelo a modelo que han surgido, se necesitan trabajos en los que se realicen comparativas de los mismos. En [78], se hace una clasificación basada en un modelo de características de las distintas propuestas para realizar transformaciones de modelos, no solamente entre

modelos, sino también de modelo a texto. En [307], también se define un *framework* para clasificar las propuestas de transformación de modelos, sin embargo, no se llega a realizar una clasificación de las propuestas existentes. Otra propuesta para clasificar las técnicas de transformación de modelos se da en [250], y este marco de clasificación se aplica más tarde en [251] para cuatro propuestas basadas en transformaciones de grafos (AGG [373], Fujaba [158], GReaT [362] y VIATRA [75]). En [369] también se comparan cuatro propuestas de transformación de grafos: AGG, AToM3 [79], VIATRA2 [390] y VMTS [378].

En cuanto a comparativas más generales, en [404] se comparan varias herramientas de transformación de modelos, mientras que en [371] y [68] se comparan herramientas desde una perspectiva MDA.

4.8.5 Herramientas de composición de modelos

Desde la perspectiva de que todo es un modelo, existen herramientas y algoritmos de composición que provienen de diferentes ámbitos. Esta sección se va a centrar exclusivamente en aquellas herramientas que provienen del ámbito del DSDM. Note que algunas propuestas definidas en el ámbito del modelado orientado a aspectos también se pueden considerar herramientas de composición, pero se han dejado fuera de esta sección porque ya se han tratado en el capítulo 3. A diferencia de lo que ocurre con la transformación de modelos, para la composición de modelos no hay ningún estándar definido, ni un vocabulario común dentro de las distintas áreas en las que hay propuestas de composición de modelos.

Una primera opción para componer modelos es definir transformaciones de modelos usando cualquiera de las herramientas presentadas en la subsección 4.8.4. En este caso, el uso de un lenguaje de transformación de modelos puede asemejarse al uso de un lenguaje de propósito general. El resto de herramientas presentadas en esta sección son semejantes a lenguajes de propósito específico, en el sentido de que proporcionan operaciones propias de la composición de modelos, aunque hay que tener en cuenta que algunas de ellas usan las transformaciones como base para implementar estas herramientas de composición de modelos. La tabla 4.6 muestra un resumen de las herramientas que se tienen en cuenta en este apartado.

Propuesta/Lenguaje	Ref	Herramienta
AMW	[83]	Plataforma AMMA [93]
EML	[221]	Plataforma Epsilon [97]
MoMENT	[43]	The MoMENT Project [159]

Tabla 4.6: Herramientas de composición de modelos

Atlas Model Weaver (AMW) [93] es parte de la plataforma de gestión de modelos AMMA, y su objetivo es definir relaciones entre elementos de distintos modelos. En esta herramienta la composición de modelos es un caso particular del *weaving* de modelos,

y es una operación que se realiza en dos fases. En primer lugar, mediante un modelo de *weaving* se definen las relaciones entre los elementos de los modelos de entrada. La creación del modelo de *weaving* requiere la intervención del usuario, aunque hay parte del proceso que se puede automatizar mediante transformaciones de modelos automatizadas, que en la propuesta se llaman transformaciones de *matching*. En la segunda fase, a partir del modelo de *weaving* se genera una transformación de forma automática, que es la que implementa la operación de composición. Al ser parte de la plataforma AMMA, esta transformación se define en el lenguaje de transformación de modelos ATL [94].

Epsilon Merging Language(EML) [96] es un lenguaje que forma parte del conjunto integrado del lenguajes definidos en el proyecto Epsilon. EML incluye un lenguaje de comparación de modelos (ECL) y un lenguaje de transformación de modelos (ETL). La composición de modelos con Epsilon también se realiza en dos fases. En la primera fase se definen reglas de comparación mediante ECL. Las reglas de comparación tienen dos partes, una de comparación y otra de conformidad. Al aplicar estas reglas, los elementos de los modelos de entradas se clasifican en elementos que tienen una correspondencia, y elementos que no tienen correspondencia. En la segunda fase, se realiza la composición, de tal forma que los elementos que tienen correspondencia se mezclan, mediante reglas de tipo *merge*, y los elementos que no tienen correspondencia se transforman mediante reglas de transformación ETL.

MoMENT [159] es un *framework* de gestión de modelos y proporciona una serie de operadores para tratar con modelos. El operador relacionado con la composición de modelos es el operador *merge*, que se define como la unión disjunta de los dos modelos de entrada, es decir, la unión de los elementos de los dos modelos de entrada eliminando los duplicados. Para realizar la operación *merge* los dos modelos a componer tienen que ser conformes al mismo metamodelo.

4.9 Sumario

En este capítulo se ha dado una visión general del nuevo paradigma conocido como Desarrollo de Software Dirigido por Modelos, incidiendo en la propuesta de la OMG. MDA ha hecho que los modelos, el metamodelado y las transformaciones se conviertan en elementos fundamentales dentro del proceso de desarrollo de software. En [332] se vio la necesidad de elevar el nivel de abstracción de los lenguajes de modelado de aspectos y de hacer a éstos independientes de la plataforma de implementación. Así, en [339] se hacía un estudio de los componentes necesarios para implementar un *framework* aplicando los principios de DSDM, mientras que en [325] se hacía un estudio de la necesidad de definir transformaciones oblicuas para realizar lo que se conoce como *model weaving*.

Finalmente, en [327] se realiza una propuesta para mejorar la extensión de los lenguajes de metamodelado aplicando ideas del desarrollo de software orientado a aspectos.

The Internet is like a giant jellyfish. You can't step on it. You can't go around it. You've got to get through it.

John Evans

5

Modelado de la Navegación

Este capítulo presenta los fundamentos de la navegación web y muestra cómo algunas propuestas, tanto orientadas al diseño, como al análisis, verificación y pruebas de sitios web se enfrentan el modelado de la navegación. Para ello, el capítulo se ha dotado de la siguiente estructura: en primer lugar, en la sección 5.1 se introduce el capítulo. Luego, en la sección 5.2 se explican los fundamentos de la navegación desde una perspectiva de la arquitectura de la información. A continuación, en la sección 5.3 se introducen las propuestas orientadas al diseño de sitios web, proporcionando en primer lugar una clasificación de las mismas, y detallando cómo se enfrentan a la navegación seis propuestas de este grupo: Web Modeling Language (`WebML`), UML-based Web Engineering (`UWE`), Object-Oriented Hypermedia Design Method (`OOHDM`), Hera-S, Object-Oriented Web Solutions (`OOWS`)y `WebDSL`. El motivo para escoger estas seis propuestas concretas es que todas ellas, además, están en proceso de adopción de algunas de las ideas del diseño del software orientado a aspectos, tal como se verá con más detalle en el próximo capítulo. En la sección 5.4 se da una clasificación de las propuestas orientadas al análisis, verificación y pruebas de sitios web. Al igual que ocurría con las propuestas orientadas al diseño, en este apartado se detallan dos propuestas: Formal Approach for Rich Navigation (`FARNav`) y Web Applications Analysis and Testing (`WAAT`), ya que también aplican ideas del `DSOA`. Finalmente, en la sección 5.5 se resume el capítulo.

5.1 Introducción

Algunos de los retos más importantes en los que se tienen que centrar los desarrolladores web no son siempre de carácter tecnológico [192], sino que están relacionados con preguntas tales como: qué se debe incluir en el sitio web, cómo se debe organizar el sitio o cómo podrán los usuarios encontrar el camino hacia lo que buscan. Estas preguntas están relacionadas con el concepto de navegación, y se han intentado responder principalmente desde dos disciplinas diferentes: la Arquitectura de la Información (`AI`) y la Ingeniería Web (`IW`). Mientras que la Arquitectura de la Información se encarga de organizar, estructurar y etiquetar la información, el diseño de la navegación en la Ingeniería Web está más relacionado con el desarrollo técnico y el diseño visual.

La forma de concebir la navegación y el objetivo que se persigue van a influir en la forma de modelarla. Así, en la bibliografía se pueden encontrar propuestas cuyo objetivo es simplificar la construcción de aplicaciones web y usan métodos de diseño descendente, y propuestas que usan ingeniería inversa para extraer modelos a partir de aplicaciones ya existentes con objeto de dar soporte a su mantenimiento y evolución. En general, las propuestas enfocadas al desarrollo de aplicaciones web usan el principio de separación de conceptos definiendo cinco perspectivas de una aplicación web: un modelo conceptual, un modelo de navegación o hipertexto, un modelo de presentación, un modelo de *workflow* y un modelo de arquitectura, especificando el modelo de la estructura de la navegación como una vista del modelo conceptual. Las propuestas cuyo objetivo es ayudar en el mantenimiento, evaluación y evolución de las aplicaciones web se apoyan más en el concepto intuitivo de pulsar un enlace para definir la navegación, y no tienen en cuenta los procesos de negocio.

Dentro del propio modelado de la navegación se han de tener en cuenta dos perspectivas: la estructural y el comportamiento. La primera tiene que ver con cómo los diferentes nodos de navegación se definen y enlazan, mientras que la segunda se centra en las acciones de navegación de los usuarios y en los eventos que generan provocando cambios cuando se atraviesan las diferentes estructuras de hipertexto.

Además de tener en cuenta estas dos perspectivas, la navegación se puede descomponer en diferentes *concerns*. Según [197], un *concern* navegacional es un *concern* que afecta a la estructura de navegación de una aplicación web, y que, por lo tanto, afecta al modo en que un usuario navega por un sitio web. Normalmente una aplicación tiene que tratar con una gran cantidad de *concerns* relacionados con la navegación. En [197] se definen tres grandes categorías de *concerns* que afectan a la navegación:

- *Concerns* de tareas. Es la categoría más amplia y recoge aquellos *concerns* relacionados con las diferentes acciones de alto nivel que un usuario puede realizar en una aplicación web, como explorar los productos, ver el carrito, proceder a la compra, etc. Algunos de estos *concerns* también están relacionados con los servicios que proporciona una aplicación web, como las recomendaciones de compra en `Amazon`.
- Temas. Este tipo de *concern* suele aparecer en sitios puramente informativos o en

el contexto de las tareas. Ejemplos de este tipo de *concern* son los temas o tópicos que aparecen en una enciclopedia, o en el contexto de una tarea, la búsqueda de un producto por su categoría mientras se realiza una compra en una aplicación de comercio electrónico.

- *Concerns* de navegación "puros". En esta categoría están los *concerns* relacionados con abstracciones que se usan en el diseño de la navegación, como por ejemplo las visitas guiadas.

Teniendo esta clasificación en cuenta, dos *concerns* relacionados con la estructura de la navegación, y que pueden clasificarse como "puros", son la navegación estructural y la de utilidad. Ambos son dos tipos de navegación reconocidos en el área de la Arquitectura de la Información. El primero sirve para estructurar un sitio web, mientras que el segundo proporciona acceso a facilidades para usar los sitios.

En relación al comportamiento de la navegación, un *concern* que aparece debido a la inclusión de tareas en el modelado de aplicaciones web son los flujos de navegación, que representan la interacción que tiene el usuario con una aplicación web. Desde el punto de vista de la interacción, existen dos tipos de aplicaciones, aquéllas en la que no es necesario mantener el estado en el que ésta está en el servidor, como ocurría en los sitios web estáticos, o aquéllas en las que es necesario mantener el estado, como ocurre en las aplicaciones web. La navegación que tiene lugar cuando no es necesario mantener información del estado se conoce como *navegación libre* y puede verse como el acceso a una serie de páginas conectadas por enlaces. Cada enlace permite acceder a un recurso público. Se dice que es navegación libre porque los usuarios tienen acceso a los enlaces de forma libre (incluso éstos pueden guardarse en `Favoritos`). Es decir, no hay ningún flujo de página controlado, ni hay una tarea a completar. Si, por contra, hay que mantener el estado, se habla de un *flujo de páginas controlado*, que no es más que una tarea de usuario formada por una serie de pasos, con un punto de partida y un punto final. En este caso, lo que es accesible como recurso público es solamente el punto de partida. De esta forma, muchos autores separan la navegación controlada en lo que llaman tareas o procesos de negocio. Un ejemplo de secuencia de navegación libre sería ver un catálogo de productos, y desde él acceder a los detalles de un producto concreto, mientras que un ejemplo de flujo controlado puede ser completar una secuencia de pasos para hacer la reserva de un vuelo. Normalmente, la navegación de una aplicación web está formada por un conjunto de secuencias de navegación libres, semejantes a la de los sitios web estáticos y una serie de flujos de páginas controlados.

5.2 Fundamentos de la navegación

Esta sección presenta los fundamentos de la navegación desde la perspectiva de la arquitectura de la información, una de las disciplinas relacionadas con el diseño de la

navegación de los sitios web. Estos fundamentos están tomados de [206], y están relacionados con los elementos necesarios para diseñar la navegación y con el comportamiento de la misma. En primer lugar, en el apartado 5.2.1 se especifican los mecanismos de navegación, que son grupos de enlaces que tienen un comportamiento y un aspecto similar. Los mecanismos de navegación utilizados para diseñar un sitio concreto van a depender de cuestiones tales como el comportamiento de los propios enlaces o el tipo de contenido al que acceden. Así, en el apartado 5.2.2, se clasifican los tipos de navegación atendiendo al objetivo de la misma, es decir, hacia dónde llevan al usuario. Y, finalmente, en el apartado 5.2.3 se clasifican los distintos tipos de página, ya que los tipos de página están muy ligados con los tipos de navegación. La figura 5.1 muestra un mapa conceptual que adelanta los conceptos que se van a tratar en estos tres subapartados.

5.2.1 Mecanismos de navegación

En [206] se define un mecanismo de navegación como un enlace o grupo de enlaces que se comportan de una manera similar y tienen una apariencia similar. Los mecanismos detectados son los siguientes:

Navegación paso a paso (*Step navigation*) Este mecanismo permite al usuario moverse secuencialmente a través de las páginas. Normalmente comprende una etiqueta de texto y una flecha, acompañada por un enlace que permite moverse a la página siguiente. Una flecha apuntando hacia la izquierda indica un movimiento a la página anterior y una flecha a la derecha indica un movimiento a la página siguiente. Este tipo de mecanismo se utiliza en procesos en los que la decisión en un paso afecta al siguiente, como por ejemplo, un proceso de compra.

Navegación paginada (*Paging navigation*) Es similar al paso de navegación anterior, pero además de los enlaces al siguiente y al anterior incluye información adicional y opciones. Normalmente, no se pueden ver todos los resultados a la vez, ya que las páginas suelen tener limitada la cantidad de información que se muestra en ellas de una vez. Una vez que se alcanza el límite, entonces se muestra otro trozo del conjunto de resultados, repitiéndose el proceso hasta que todos los resultados se representan en varias páginas. La forma más simple de navegación paginada es una navegación de un paso al que se le añade un contador de páginas. Normalmente este contador aparece entre los enlaces de atrás y adelante.

Dentro del mecanismo de navegación paginada también pueden aparecer los siguientes elementos:

- *Rewind and Fast Forward*. Algunas veces los usuarios necesitan volver directamente a la primera página o adelantarse hasta la última página del conjunto. A menudo una doble flecha, o una flecha con una barra vertical representa este tipo de navegación.

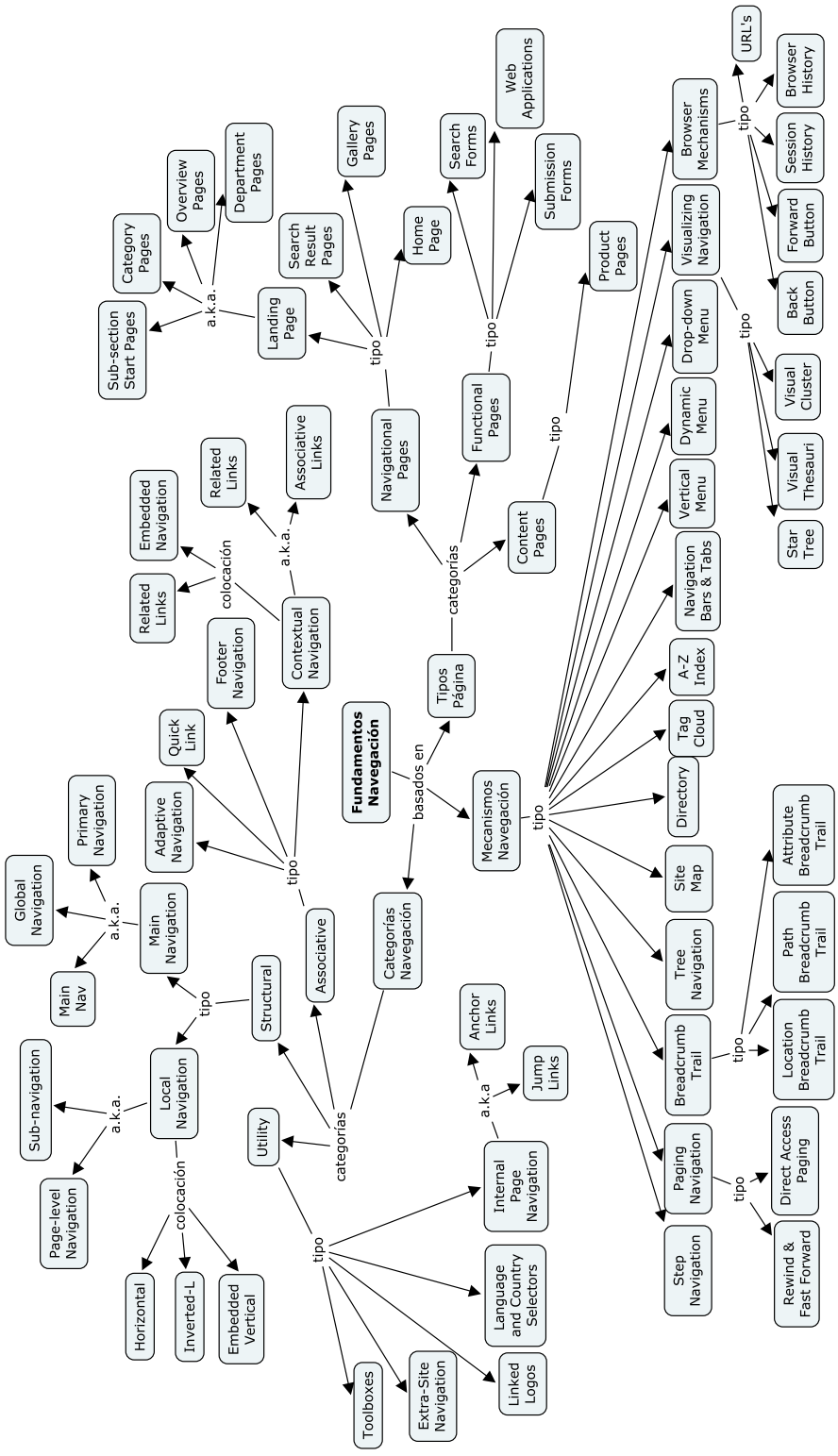


Figura 5.1: Mapa conceptual con los fundamentos de la navegación

- Paginación de acceso directo (*Direct Access Paging*). Este tipo de mecanismo se ve a menudo en la parte inferior de las páginas de los motores de búsqueda. Normalmente se muestra una cuenta lineal de las páginas (por ejemplo, página 1, 2, 3, 4, etc.) junto con los controles de navegación por pasos. De esta forma se permite el acceso directo a cualquier página del conjunto completo.

  Otra opción aquí es, en lugar de mostrar el número de páginas ir reflejando el número total de items en cada página, de tal forma de tal forma que la cuenta podría ser 1, 26, 52, etc., si asumimos que cada página muestra 25 ítems.

Rastro de miga de pan (*Breadcrumb trail*) Este mecanismo le muestra a una persona el camino que ha seguido a través de un sitio. Comprende una serie de elementos o nodos, que encadenamos juntos. Los nodos se enlazan a páginas visitadas previamente (o temas padres) y se separan con un símbolo, normalmente un mayor (>), dos puntos (:) o una barra vertical (|).

Existen tres tipos de rastro de miga de pan:

- Rastro de miga de pan de localización (*Location breadcrumb trails*). Este tipo normalmente muestra la posición actual dentro de un sitio y proporciona atajos hacia páginas vistas previamente y/u otras áreas del sitio. En esencia, este tipo de rastro de miga de pan es una representación lineal de la estructura del sitio.

- Rastro de miga de pan de ruta (*Path breadcrumb trails*). Este tipo de rastro de miga de pan es dinámico, en el sentido de que cualquier página mostrará un rastro de miga de pan distinto, ya que éste está basado en la ruta que ha seguido el usuario para alcanzar la página.

- Rastro de miga de pan de atributo (*Attribute breadcrumb trails*). Describe una página de alguna manera, en lugar de mostrar su localización dentro de un sitio o la ruta para llegar hasta allí. Muestran su posición dentro de un esquema de metadatos, a menudo una jerarquía de temas.

Navegación en árbol (*Tree Navigation*) Este tipo de navegación permite el acceso a una estructura jerárquica. Se ve a menudo en sistemas operativos para navegar por las carpetas de directorios. Normalmente se muestra como una disposición vertical de carpetas, términos o nodos pertenecientes a alguna jerarquía. A menudo hay un símbolo pequeño de suma (+) y un símbolo de menos (-) para abrir o cerrar los nodos de la jerarquía. También puede haber pequeñas flechas que apunten hacia la derecha y hacia abajo para representar que un nodo está abierto o cerrado, respectivamente.

Mapas del sitio (*Site Maps*) Un mapa del sitio es una representación de la estructura del sitio web usada para la navegación. Esto proporciona una visión de arriba hacia abajo y de un vistazo del contenido del sitio. Utilizando este mapa, los usuarios pueden saltar directamente a cualquier página listada.

Directorios (*Directories*) Un directorio normalmente proporciona acceso a las páginas a través de temas. La principal diferencia entre los directorios y los mapas del sitio es que los primeros pueden clasificar el contenido por categoría. También son diferentes de los índices, que listan los términos de forma alfabética. Los directorios son útiles cuando se trata con distintos tipos de información mezclados sin ninguna relación jerárquica. También son efectivos para enlazar a sitios externos.

Nubes de etiquetas (*Tag clouds*) En este mecanismo los enlaces aparecen listados alfabéticamente y se les asocia un peso relacionado con la frecuencia, de forma que cuanto más frecuente sea la aparición del tema, mayor es el tamaño de la etiqueta que lo muestra. Con esto se da una foto instantánea de la importancia relativa de un tema.

Este tipo de mecanismo es bueno para sitios de contenido dinámico, pero como mecanismo de navegación tiene un valor limitado, ya que si un usuario busca una información previamente conocida, realmente no es de gran utilidad.

Índice A-Z (*A-Z Index*) Es una guía alfabética a los temas, términos y conceptos que se encuentran en todo el sitio web. Los índices son normalmente un punto suplementario de acceso al contenido y no son un punto de entrada principal. Proporcionan una vista de arriba hacia abajo del contenido del sitio.

Barras y pestañas de navegación (*Navigation Bars and Tabs*) La forma más simple de una barra de navegación es una cadena horizontal de enlaces de hipertexto. Algunas veces se separan por una línea vertical. La distinción entre las barras y las pestañas es la presentación, ya que realmente no hay diferencia en su funcionamiento. En lugar de una barra sólida las pestañas normalmente se colocan encima del área de contenido de la Web. Tanto las barras como las pestañas tienen problemas de escalabilidad, ya que las páginas tienen un ancho predefinido.

Menú vertical (*Vertical Menu*) Es una lista de enlaces que normalmente se sitúa en la parte izquierda o la parte derecha de la página y que se mantiene en todo el sitio web. Este mecanismo es más flexible que las barras de navegación o las pestañas porque se puede extender fácilmente gracias a la adicción de nuevos items al menú por la parte inferior del mismo.

Menú dinámico (*Dynamic Menu*) También se conocen como menús flotantes. Este tipo de menús proporcionan un acceso rápido a las opciones de navegación. Se consideran dinámicos porque el usuario debe interactuar con ellos antes de mostrarlos. Después de que el usuario selecciona una opción de navegación con el ratón o con un click en el mismo, se le presenta una ventana de menú similar a las encontradas en las aplicaciones software ordinarias. Una ventaja clave de este mecanismo es el acceso a más opciones de las que podrían ser mostradas en una página.

Menú desplegable (*Drop-Down Menu*) Este tipo de menú son menús de selección simples de `HTML` con opciones. Seleccionar una opción lleva al usuario a una nueva página. Este tipo de navegación normalmente se utiliza para enlaces rápidos, que llevan a una página nueva de la estructura del sitio.

Navegación de visualización (*Visualization Navigation*) En este grupo se encuentran una serie de mecanismos que usan información de visualización. En este área la información se representa mediante relaciones visuales o espaciales para lograr que conjuntos de datos complejos sean claros y se puedan entender bien. Además, estas representaciones son interactivas. Este tipo de información no sustituye a la información textual, sino que se mezcla con ella para complementarla. Los tres mecanismos más comunes usados con la visualización de información son:

- Árbol en Estrella (*Star Tree*). También se le llama distribución en árbol radial y representa relaciones jerárquicas en forma radial (*hub and spoke*). De esta forma se pueden representar grandes cantidades de datos en un espacio relativamente pequeño. Un árbol en estrella debería ser una forma alternativa a un árbol de navegación e incluso a un mapa del sitio.

- Tesauro Visual (*Visual Thesauri*). Se suele representar el tema principal de la página en el centro y los términos relacionados flotando alrededor de él. A diferencia de los árboles en estrella que pretenden mostrar una gran cantidad de información en un espacio pequeño, el objetivo de un tesauro visual es reforzar la exploración del usuario. Generalmente muestra un conjunto limitado de conceptos relacionados para ayudar en el descubrimiento de información nueva.

- Agrupaciones Visuales (*Visual Clusters*). Este tipo de mecanismo se está aplicando a la visualización de resultados de búsqueda. Los resultados dentro de una categoría similar se agrupan dentro de círculos. Se puede hacer zoom para acercarse o alejarse y explorar así cada categoría de círculos. Las páginas web se muestran como pequeños iconos, de forma que cuando se pasa el ratón por encima de ellas se muestran más detalles de las mismas.

Mecanismos del Navegador (*Browser Mechanisms*) Finalmente, hay que resaltar que los navegadores tienen incorporados algunos mecanismos de navegación. Los más importantes son:

- Botón Atrás (*Back Button*). Permite navegar a la página visitada anteriormente.

- Botón Adelante (*Forward Button*). Es una forma de dar un paso adelante en un camino que ya se ha visitado previamente.

- Historial de la sesión (*Session history*). Es una lista cronológica en orden inverso de las páginas visitadas más recientemente.

- Historial del navegador (*Browser history*). Es un historial de las páginas visitadas que se puede remontar a semanas atrás.
- `URL`. Mediante este mecanismo también se navega a una página concreta introduciendo la dirección de la misma en el cuadro de texto destinado a tal efecto.

5.2.2 Categorías de navegación

Existen tres grandes categorías primarias o tipos de navegación, si atendemos al objetivo de la misma, es decir, hacia dónde lleva al usuario [120, 206, 413]: estructural, asociativa y de utilidad. En la figura 5.2 se muestra un esquema de estas categorías primarias, las cuales se detallan a continuación:

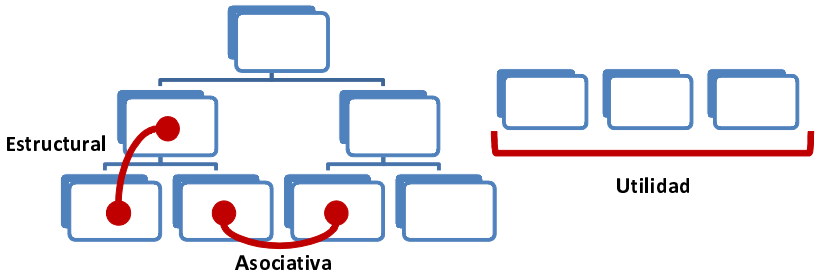


Figura 5.2: Categorías primarias de navegación según Fiorito y Dalton [120]

Estructural (*Structural*) Con este tipo de navegación se conectan las páginas basadas en la jerarquía del sitio. En cualquier página esperas poderte mover a la página que está situada por encima o por debajo en la jerarquía. La navegación estructural se puede dividir en dos tipos:

- Navegación Principal (*Main navigation*). Representa a las páginas situadas a más alto nivel en la jerarquía del sitio web, o las páginas situadas justo debajo de la página *home* del sitio. Los enlaces que representan la navegación principal suelen aparecer en una posición consistente y que no cambia en casi todas las páginas de un sitio.
- Navegación Local (*Local navigation*). Se usa para acceder a los niveles más bajos de la jerarquía del sitio, por debajo de las páginas de navegación principal. Dada una página, la navegación local generalmente muestra otras opciones al mismo nivel de la jerarquía, así como las opciones por debajo de la página actual. Normalmente la navegación local trabaja en conjunción con la navegación principal, y aunque puede verse como una extensión de la navegación principal, normalmente se trata de forma diferente porque varía más a menudo.

Las distribuciones comunes para la navegación local y principal son:

- L Invertida (*Inverted-L*). Es común colocar la navegación principal a lo largo de la parte superior de la página y tener la navegación local como una lista de enlaces en la parte izquierda de la página, tomando forma de L invertida.
- Horizontal (*Horizontal*). La navegación local se representa como una segunda fila de opciones situada por debajo de una navegación principal horizontal, o bien por menús dinámicos.
- Vertical embebida (*Embedded Vertical*). La navegación principal se presenta en un menú, bien a la izquierda, bien a la derecha, y la navegación local se embebe entre las opciones de la navegación principal tomando forma de una estructura de árbol.

Asociativa (*Associative*) Conecta páginas con temas y contenidos similares; en lugar de su localización en el sitio, los enlaces tienden a cruzar la estructura jerárquica del mismo. Principalmente existen tres tipos de navegación asociativa:

- Navegación Contextual (*Contextual Navigation*). La navegación contextual puede variar, depende de la situación. Con frecuencia, los enlaces pueden conducir a nuevas áreas de contenido, tipos de páginas diferentes, o incluso a un nuevo sitio. Normalmente la navegación contextual se coloca cerca del contenido de una página, creando así una fuerte conexión entre el significado del texto y el conjunto de páginas relacionadas. Normalmente se distribuye de dos formas: embebida en el texto (*Embedded navigation)*, al final o al lado del contenido (*Related links*).
- Enlaces rápidos (*Quick Links*). Los enlaces rápidos dan acceso a contenido o áreas importantes del sitio web que pueden no estar representadas en una navegación principal. Aunque es similar a la navegación contextual, los enlaces rápidos son contextuales para el sitio entero, no para una página dada. Generalmente resaltan áreas de contenido o tareas a las que se accede frecuentemente y a menudo se colocan en la parte superior o en los laterales de la página.
- Navegación de pie de página (*Footer Navigation*). Normalmente se representa mediante enlaces textuales que se colocan al final de la página. A menudo, acceden a páginas simples sin niveles de estructura por debajo de ellas. Normalmente la navegación de pie de página contiene información que no está relacionada con el tema principal del sitio, tal como el copyright, los términos y condiciones, y los créditos del sitio.

Utilidad (*Utility*) Conectan páginas y características que ayudan a los usuarios a usar el sitio. Normalmente permanecen fuera de la jerarquía del sitio. El mecanismo de navegación usado para representar los enlaces de navegación de utilidad suele ser más

pequeño que que los mecanismos usados para representar la navegación principal, apareciendo en la parte superior, los laterales o la parte inferior de la navegación principal. Existen varios tipos de navegación de utilidad:

- Cajas de herramienta (*Toolboxes*). Coloca juntas opciones del sitio que realizan funciones (herramientas) para hacer cosas en el sitio. Pueden contener enlaces al contenido o a páginas de navegación, pero a menudo enlazan con páginas funcionales.
- Logos enlazados (*Linked logos*). Suele ser un logo que se coloca en la parte superior de la página. Es habitual que se enlace la imagen a la página raíz del sitio.
- Selectores de idiomas y países (*Language and Country Selectors*). Esta categoría de navegación se utiliza en sitios que tienen versiones en distintos idiomas. Normalmente se utiliza un selector de idioma, que le permite al usuario cambiar a las distintas versiones del sitio. Una variante de esta categoría es usar un selector de país o región de mercado, ya que el contenido puede variar de un país a otro.
- Navegación de páginas interna (*Internal Page Navigation*). Esta categoría se utiliza cuando las páginas son muy grandes y es, por lo tanto, interesante añadir enlaces internos que permitan saltar de una sección a otra dentro de la misma página, básicamente este tipo de enlaces permiten hacer *scroll* hacia arriba y hacia abajo en una página de forma eficiente.

5.2.3 Tipos de páginas

En [206] clasifica las navegación en relación al objetivo de la misma en tres grandes categorías:

Páginas navegacionales (*Navigational pages*) El objetivo de las páginas navegacionales es indicarle al usuario el camino hacia su objetivo final, ya sea una página de contenido o una funcional. Algunas páginas navegacionales importantes son la página principal o *home* del sitio, páginas de inicio, galerías o páginas de resultados de búsqueda.

- Página principal (*Home Page*). Son enlaces a otros sitios relacionados, subsitios o compañías. Normalmente se utiliza en sitios de grandes corporaciones que pueden tener diversas áreas de productos o negocios.
- Página de inicio (*Landing Page*). Las páginas de inicio proporcionan una visión general de las principales categorías del sitio web, que a menudo corresponden a las opciones en una navegación principal. La principal diferencia entre una página principal y una página de inicio es que mientras la página principal da una visión general de todo el sitio web, la página de inicio proporciona un resumen del contenido de una sección dada.

- Página de galería (*Gallery Page*). Son similares a las páginas de inicio, pero en lugar de proporcionar enlaces para un departamento o una sección del sitio, da una visión general de un grupo específico de productos o contenido.
- Página de resultados de búsqueda (*Search Result Page*). Son similares a las páginas de inicio, pero en lugar de proporcionar enlaces para un departamento o una sección del sitio, dan una visión general de un grupo específico de productos o contenido.

Páginas de contenido (*Content pages*) En los sitios web ricos en información las páginas web son lo que el usuario va buscando en último lugar. El centro de estas páginas debería ser el contenido, aunque muchas veces en la Web nos encontramos con páginas de contenido llenas de navegación y gráficos innecesarios. Un tipo de página de contenido esencial para los sitios web de comercio electrónico son las páginas de productos.

Páginas funcionales (*Functional pages*) Permiten al usuario completar una tarea, tal como realizar una búsqueda o enviar un e-mail. Suelen ser páginas que tienen poco texto y no suelen contener ni navegación embebida ni enlaces relacionados.

- Formularios de búsqueda (*Search Forms*). Es bastante común que la característica de búsqueda del sitio sea un pequeño cuadro de entrada de texto en la página principal. Algunas veces se necesita una búsqueda más detallada a la que se accede a través de un enlace de búsqueda avanzada. En este tipo de formularios no suele haber navegación, más allá de algún botón de ayuda.
- Formularios de envío (*Submission Forms*). Permiten al usuario el envío de información. Al igual que ocurría con los formularios de búsqueda, no es común encontrar navegación asociativa.
- Aplicaciones web (*Web Applications*). Son un rango de páginas que contienen funcionalidad y características interactivas. Los usuarios realizan tareas en estas páginas: envían correos electrónicos, editan hojas de cálculo, gestionan proyectos, etc. Las aplicaciones de correo electrónico son un tipo común de aplicación.

5.3 Modelado de la navegación en propuestas orientadas al diseño de sitios web

En general, las propuestas orientadas al diseño de sitios web definen cinco actividades principales a la hora de enfrentarse al diseño de un sitio [55]: el diseño de datos o modelo conceptual, el diseño de *workflows*, el diseño de la navegación, el diseño de la presentación y el diseño de la arquitectura. Las dependencias entre estas actividades se muestran en la figura 5.3.

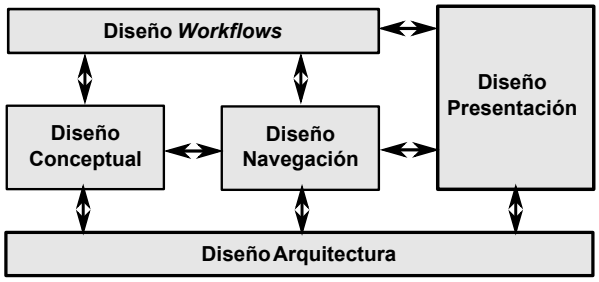


Figura 5.3: Actividades involucradas en el diseño web y sus dependencias según [55]

La forma en que se secuencian estas actividades ha dado lugar a dos tipos de proceso de desarrollo, dependiendo de si hace desde una perspectiva de la información o del usuario:

- Procesos de diseño centrados en la información. Las propuestas de este tipo comienzan con las actividades relacionadas con el *back-end* del sistema de información, siendo normalmente el inicio el modelado conceptual. Luego, siguen con el diseño de la navegación, y derivan los *workflows* de los artefactos producidos por las dos actividades anteriores. A continuación siguen con el diseño de la presentación y terminan con el diseño de la arquitectura.

- Procesos de diseño centrados en el usuario. En este tipo de propuestas se comienza por un análisis de las actividades del usuario, siendo el primer paso la definición de los *workflows*. El resultado se toma como entrada para el diseño de la navegación, donde los *workflows* se transforman en secuencias de pasos de navegación. Para cada paso de navegación se realiza un diseño de presentación, y en paralelo, se realiza el diseño del modelo conceptual para especificar los datos necesarios para cada una de las actividades del usuario.

Aunque la mayoría de las propuestas de esta categoría mantiene vistas separadas para el modelado conceptual, de navegación y de presentación, casi todas definen una notación específica, que depende muchas veces del origen y el objetivo de la misma, así, los métodos de modelado, se pueden clasificar en [410, 356]:

Métodos orientados a datos Son métodos que nacen del campo de las bases de datos, por lo tanto, se basan en conceptos del modelo Entidad-Relación para modelar el nivel de hipertexto. El objetivo de estos métodos es el desarrollo de aplicaciones web dirigidas por una base de datos. En este grupo se encuentran `RMM` [194], Hera [187] y `WebML` [58].

Métodos orientados a hipertexto Son métodos que se centran en el carácter de hipertexto de las aplicaciones web. Su origen proviene del campo de las aplicaciones

de hipertexto. En este grupo se encuentran HDM [136], W2000 [26], HDM-Lite [133] y WSDM [80].

Métodos orientados a objetos Están basados, bien en OMT, bien en UML. En este grupo se encuentran propuestas como OOHDM [354], UWE [217], OOWS [300] y OO-H [145].

Métodos orientados al software Son aquéllos que entienden el desarrollo de aplicaciones web desde el punto de vista de la ingeniería del software tradicional. En este grupo están WAE y WAE2 [66].

Métodos orientados a la Ingeniería Dirigida por Modelos Son métodos que siguen una filosofía de desarrollo de software dirigido por modelos. En este grupo están Netsilon [265], MIDAS [53], WebSA [245], Webile [81] y WebDSL [392].

Cuando se diseña la navegación, se tienen que tratar dos aspectos importantes: la estructura y el comportamiento. La estructura se centra en cómo se definen los diferentes nodos de navegación y cómo se enlazan para formar el hipertexto. El comportamiento se centra en las acciones de navegación del usuario y en los eventos que generan. Dolog y Bieliková [87] afirman que los modelos de comportamiento son cruciales en las aplicaciones hipermedia, pero que también los modelos estructurales son un vehículo efectivo para expresar una vista particular en un sistema hipermedia. Así que cada vez hay más propuestas que incorporan los dos tipos de modelos.

En los siguientes apartados se muestra cómo distintas propuestas se enfrentan al modelado de la navegación tanto desde un punto de vista estructural como desde un punto de vista del comportamiento.

5.3.1 Web Modeling Language (WebML)

Web Modeling Language (WebML) [58] es una propuesta que, por razones históricas, proviene del mundo de las bases de datos, y como consecuencia, sus modelos descansan sobre un modelo estructural de datos. Desde el momento de su concepción en 1998, WebML ha ido evolucionando. La figura 5.4 muestra un mapa conceptual que se puede usar como guía en el resto de la sección y que contiene los principales conceptos manejados en WebML y las relaciones entre los mismos.

La metodología que acompaña al lenguaje WebML propone un ciclo de vida inspirado en el ciclo de vida en espiral con varias fases, entre las que destaca el modelado conceptual. El modelado conceptual consiste en la definición de esquemas conceptuales, que expresan la organización de la aplicación a un alto nivel de abstracción, independientemente de los detalles de implementación, y está compuesto por dos actividades, el diseño de datos y el diseño de hipertexto.

El modelo de datos de WebML define la estructura de datos (entidades, relaciones y contenidos de datos) y está basado en el modelo Entidad-Relación. Sobre él se apoya el modelo de hipertexto, que se define mediante una notación propia. En el modelo de hipertexto de WebML se reflejan dos cosas, por una parte, la navegación, y por otra,

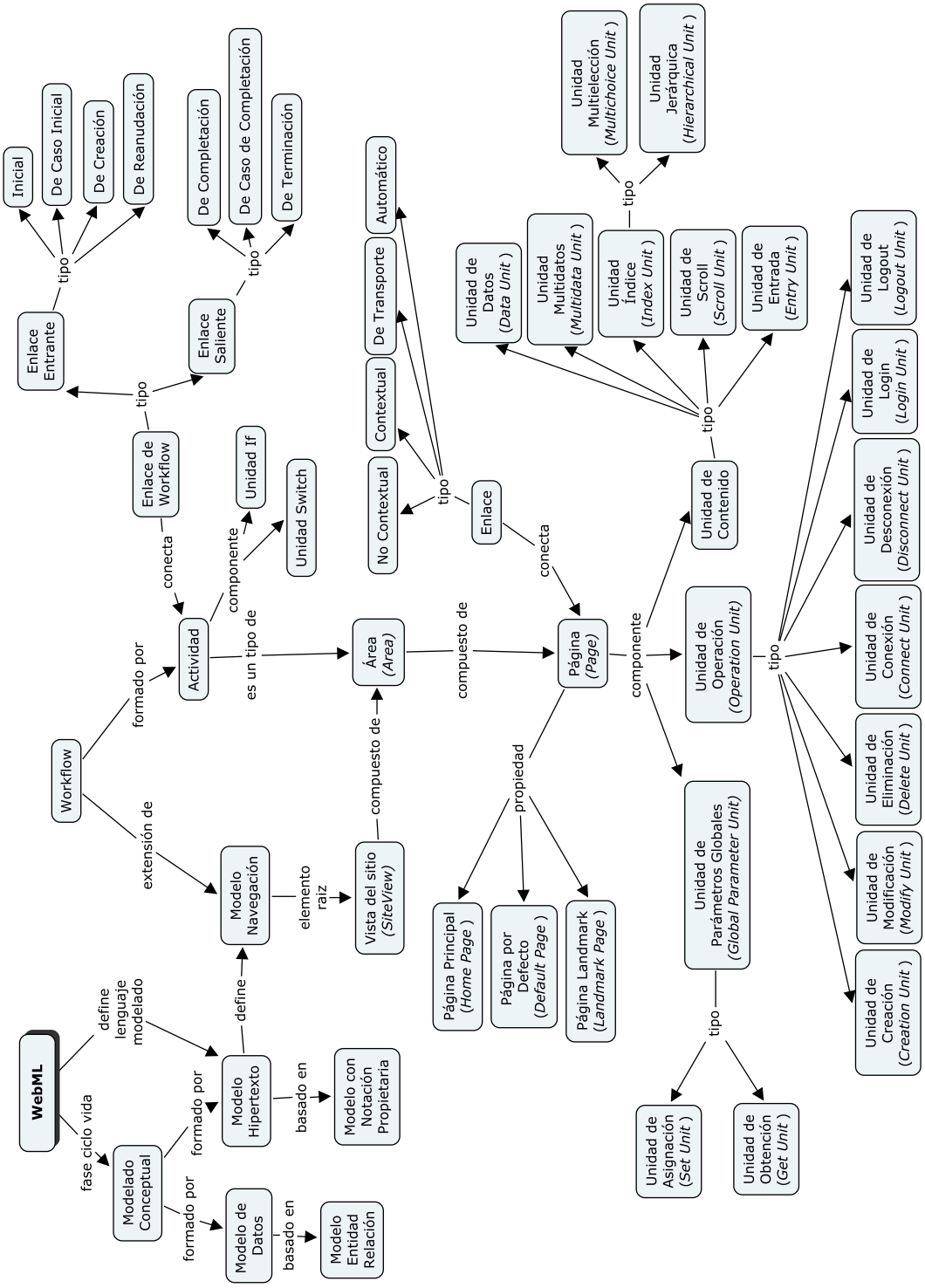


Figura 5.4: Mapa conceptual con la terminología usada en la propuesta WebML

las características de composición de las interfaces de hipertexto. El modelado de la navegación en esta propuesta es la parte del modelado de hipertexto que trata con la especificación de los enlaces entre unidades y páginas y con las propiedades de estos enlaces.

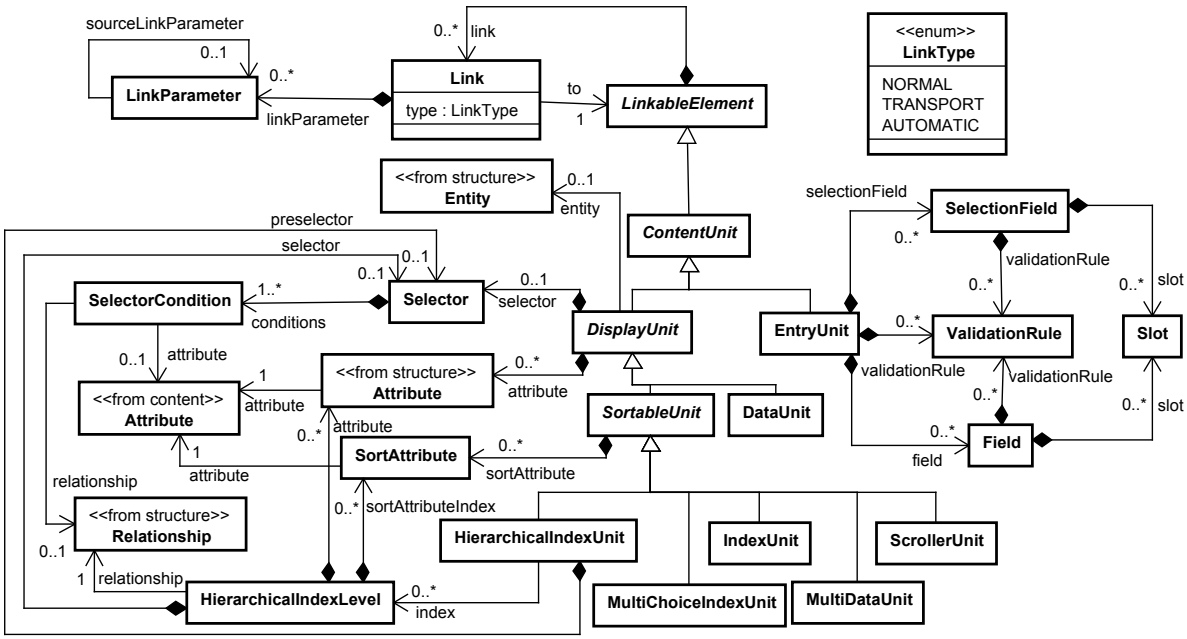


Figura 5.5: Paquete de hipertexto del metamodelo de WebML

Para el modelado del hipertexto los autores proponen una serie de primitivas: *páginas*, *unidades*, y *enlaces*, que se modularizan mediante los constructores *áreas* y *vistas de sitio*. Una *vista de un sitio* agrupa a un conjunto de áreas y representa un hipertexto que sirve a un conjunto de requisitos bien definidos. Las *páginas* se construyen ensamblando unidades de varios tipos. Las *unidades* son las partes atómicas de contenido publicable y ofrecen, tanto distintas formas de gestionar el contenido extraído dinámicamente de las entidades y relaciones del esquema de datos, como la especificación de formularios de entrada de datos. Se puede decir, por tanto, que las unidades son los bloques con los que se construyen las páginas, que son las interfaces que se le presentan al usuario. Las páginas y las unidades se enlazan para formar la estructura de hipertexto, de tal forma que los *enlaces* representan, por una parte, la posibilidad de navegar de un sitio a otro en el hipertexto y, por otra, el paso de parámetros de una unidad a otra. La figura 5.5 muestra el paquete `Hypertext` del metamodelo de `WebML` definido en [347]. En este paquete se definen los elementos relacionados con el modelado de la navegación. Como se puede observar, en este paquete no aparecen los constructores área, página o vista de sitio, ya que éstos representan la estructura de hipertexto, y se recogen el en paquete `HypertextOrganization`.

Finalmente, las unidades y las páginas se conectan mediante enlaces. Los enlaces entre unidades se denominan *contextuales*, ya que llevan información de la unidad origen a la unidad destino. Por contra, los enlaces entre páginas se llaman *no contextuales*. En los enlaces contextuales la ligadura entre la unidad origen y la destino se representa formalmente mediante los parámetros del enlace y selectores paramétricos. Los primeros están asociados con el enlace y modelados mediante la metaclase `LinkParameter`, mientras que los segundos se asocian a la unidad destino.

En `WebML` además se definen otros dos tipos de enlaces: automáticos y de transporte. Los *enlaces automáticos* se navegan sin necesidad de interacción con el usuario en el momento en el que se accede a la unidad origen que lo contiene. Los *enlaces de transporte* son enlaces que transportan información contextual y que no se representan como un ancla.

Todos los conceptos definidos en `WebML` tienen una sintaxis gráfica y una sintaxis textual.

5.3.2 UML Web Engineering (UWE)

UML-based Web Engineering (`UWE`) [217] es una propuesta de las clasificadas como método orientado a objetos. Se originó con la idea de realizar el modelado de aplicaciones web como una extensión de `UML`, centrándose en el modelado visual, en el diseño sistemático y la generación automática. Actualmente, su objetivo es cubrir todo el ciclo de desarrollo de aplicaciones web proporcionando técnicas y notaciones que cubran todo el ciclo de vida, comenzando con modelos de requisitos, siguiendo con los modelos de diseño e incluyendo modelos de arquitecturas y aspectos. La figura 5.6 muestra un mapa conceptual como guía para el resto de la sección conteniendo los principales conceptos usados en la propuesta y las relaciones entre los mismos.

Se puede decir que `UWE` es una metodología compuesta por una notación y un método que están en constante evolución. La notación está basada en un perfil de `UML` [32, 181, 226], donde se utilizan cinco tipos de diagramas: *diagrama de casos de uso*, para capturar los requisitos del sistema; *diagrama conceptual*, para el modelo de dominio; *diagrama de estructura de navegación*, para modelar los caminos de navegación de la aplicación que se está modelando; *diagrama de presentación*, para modelar la apariencia visual y, finalmente, *el modelado de tareas o procesos*. El modelado de tareas es el resultado de una extensión para tratar con el modelado de lo que en la introducción se ha denominado navegación controlada. En general, las extensiones definidas en el perfil representan los aspectos estructurales de las distintas vistas (conceptual, navegacional y presentación), aunque los autores se apoyan en diagramas de interacción y diagramas de estado `UML` para modelar la perspectiva de comportamiento del sistema web.

El objetivo del modelo de navegación es especificar la navegabilidad por el contenido de una aplicación web, es decir, definir una vista de navegación estática del contenido. Las clases de navegación representan nodos navegables de la estructura de hipertexto y los enlaces de navegación muestran rutas directas de navegación entre nodos. En `UWE`

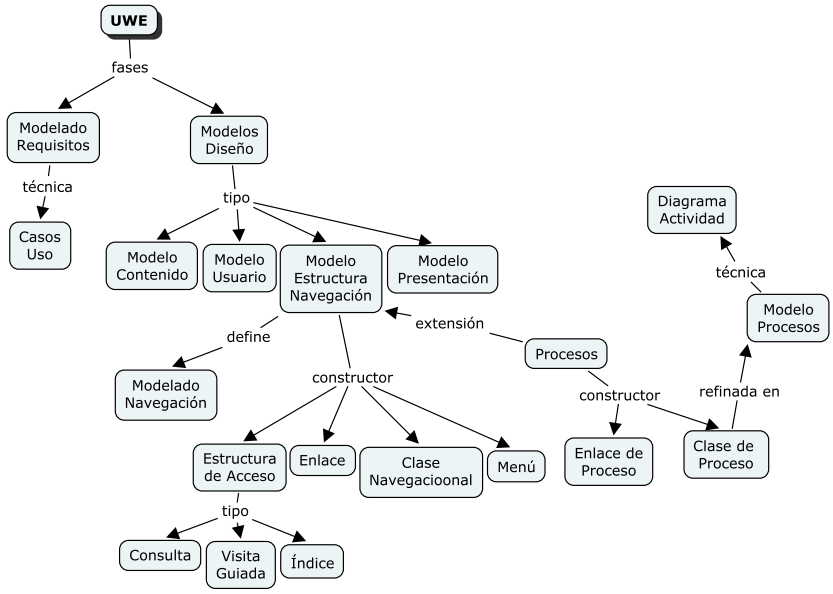


Figura 5.6: Mapa conceptual con la terminología usada en la propuesta UWE

un nodo se define como una especialización de una clase `UML`, y un enlace como una especialización de una asociación (ver figura 5.7). El modelo de navegación se construye en varios pasos: en primer lugar, se construye lo que se denomina un modelo del espacio de navegación, que indica qué nodos se pueden visitar mediante navegación directa a partir de otros nodos; en segundo lugar, se añaden a este modelo las estructuras de acceso, que indican cómo se accederá a estos objetos; finalmente, se añaden los menús.

En `UWE` se utilizan dos diagramas para modelar la navegación: un diagrama del espacio de navegación, que indica qué objetos se pueden visitar en la navegación de la aplicación, y un diagrama de estructura de la navegación, que indica cómo se accederá a estos objetos.

Los elementos centrales del paquete de navegación en `UWE` (figura 5.7) son los enlaces de navegación (`Link`) y los nodos (`Node`). Ambos están definidos como clases abstractas, lo que implica que no puede crearse ninguna instancia de estas metaclases. El atributo `isHome` de `Node` modela el hecho de que un nodo sea el punto de entrada a la aplicación, mientras que el atributo `isLandmark` indica que el nodo es alcanzable desde cualquier otro nodo del grafo de navegación. Finalmente, el atributo `isAutomatic` de la metaclase `Link` representa el hecho de que el enlace no necesita la intervención del usuario para ser lanzado automáticamente. Los enlaces conectan un nodo origen con uno o varios nodos destino (lo que se modela mediante las dos relaciones de las metaclases `Node` y `Link`). Con esta relación se extiende la semántica del enlace `HTML`, ya que en `HTML` un enlace conecta dos páginas, un origen con un destino, pero no varias.

Un nodo puede ser una clase de navegación, un menú, una estructura de acceso o una clase de proceso. Una clase de navegación (`NavigationClass`) es una vista de

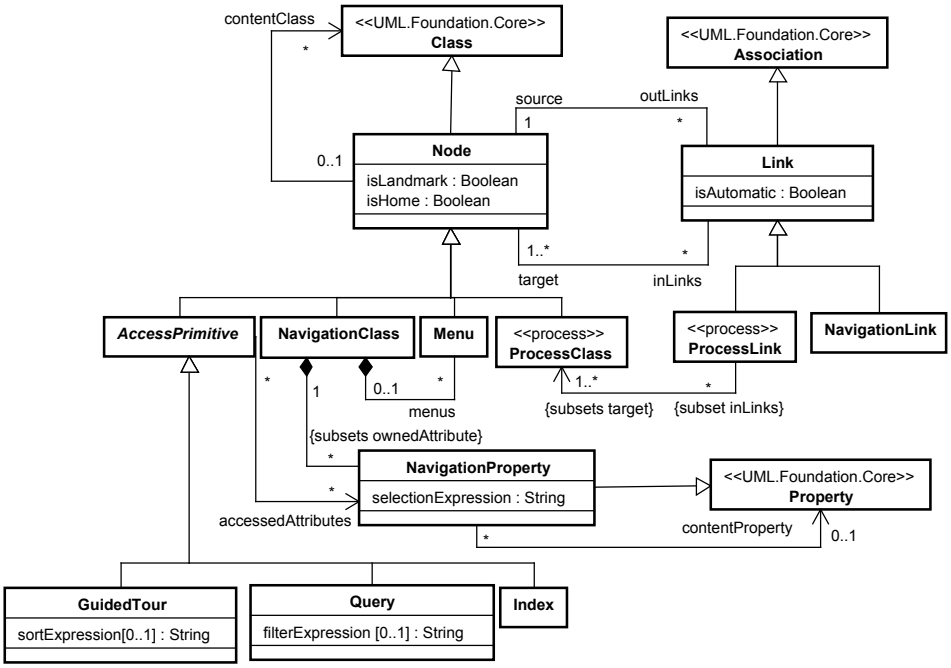


Figura 5.7: Paquete Navigation del metamodelo de UWE y sus relaciones con el metamodelo de UML tomado de [226]

una clase conceptual. Las clases de navegación están compuestas por propiedades de navegación (`NavigationProperty`), que también se derivan de la metaclase `Property` de `UML`. El menú (`Menu`) sirve para definir rutas de navegación alternativas. La estructura de acceso se modela mediante la clase abstracta `AccessPrimitive` y sirve para escoger las instancias de las clases del modelo de contenido que formarán parte de la clase de navegación. En `UWE` se definen tres primitivas de acceso: índices, consultas y visitas guiadas. Por otra parte, la clase de proceso (`ProcessClass`) se utiliza para integrar un proceso de negocio en el modelo de navegación.

Finalmente, en `UWE` se definen dos tipos de enlaces: el enlace de navegación (`NavigationLink`), que modela la navegación estática con la semántica usual de las aplicaciones hipermedia; y el enlace de proceso (`ProcessLink`), que relaciona una clase de proceso con un nodo de navegación.

Una vez que se tiene el diagrama de estructura de navegación, se puede extender con clases de procesos que representan puntos de entrada y de salida a procesos de negocio. Cada clase incluida en el modelo de navegación se refina en un modelo de proceso que consiste en un modelo de flujo de procesos y, opcionalmente, un modelo de estructura de procesos. Para modelar los flujos de proceso se utilizan diagramas de actividad `UML`.

5.3.3 Object Oriented Hypermedia Method (OOHDM)

La propuesta Object-Oriented Hypermedia Design Method (OOHDM) [354] también forma parte de grupo de las orientadas a objetos. A modo de resumen, en la figura 5.8 se presenta un mapa conceptual con los términos manejados en esta propuesta y sus relaciones.

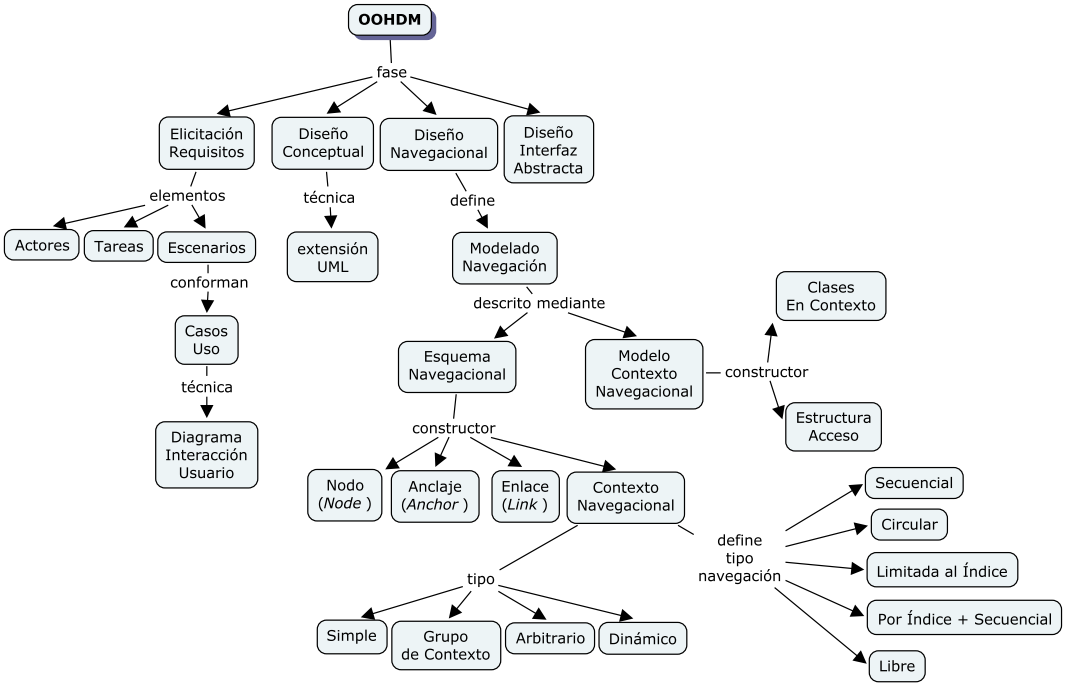


Figura 5.8: Mapa conceptual con los principales conceptos de OOHDM

En OOHDM, el espacio de desarrollo se particiona en cinco actividades: elicitación de requisitos, diseño conceptual, diseño navegacional, diseño de interfaz abstracta e implementación. Para elicitar los requisitos, OOHDM se basa en escenarios, que se capturan mediante Diagramas de Interacción de Usuarios (una forma especial de los casos de uso). En el diseño conceptual se describen las clases de la aplicación y sus relaciones utilizando UML. Para cada perfil de usuario se define una estructura navegacional diferente de acuerdo a las tareas que los distintos tipos de usuario deben realizar. El esquema navegacional define la estructura de enlaces de la aplicación web, que se construye a partir de metaclases tales como nodos, enlaces, anclas y estructuras de acceso. Cada nodo se define como una vista definida sobre objetos conceptuales, actuando como un observador de esos objetos. La separación entre los objetos y sus vistas permite personalizar la estructura de los nodos y la topología de enlaces a las necesidades del perfil de usuario correspondientes a la tarea a realizar. A los objetos del modelo de navegación se les llama nodos. Los atributos de los nodos se construyen a partir de los atributos

de las clases conceptuales mediante un lenguaje de consulta. Los enlaces también se definen a partir de las relaciones del modelo conceptual.

Además del esquema de clases navegacionales derivado del modelo conceptual, en OOHDM se definen los esquemas de contextos navegacionales, que se integran con las clases de navegación y que definen tanto los conjuntos de nodos que el usuario atravesará como la topología de navegación intra-conjunto. Los contextos de navegación se pueden ver como una forma de estructurar el espacio de navegación. La estructuración se consigue mediante la agrupación de varios objetos de navegación, que es lo que se conoce como contexto de navegación. Los contextos de navegación se representan utilizando una notación propia, y los autores de OOHDM los han clasificado en los siguientes tipos:

Contextos simples Con este tipo de contexto se seleccionan todos los objetos de una clase que cumplen una propiedad relacionada directamente con esa clase (por ejemplo, profesores con categoría asociado), o todos los objetos relacionados con un objeto dado a partir de un enlace (por ejemplo, cursos impartidos por un determinado profesor).

Grupos de contextos Define grupos de contextos derivados de una clase, en la que se parametriza la propiedad que define el contexto (por ejemplo, profesores por categoría) o grupos de contextos derivados de un enlace, en cuyo caso el grupo se obtiene variando el elemento origen del enlace (por ejemplo, cursos enseñados por profesor).

Contexto arbitrario En este caso, los elementos del contexto se escogen explícitamente por el autor del contexto, pudiendo pertenecer a clases de navegación diferentes. Las visitas guiadas pertenecen a este tipo de contexto.

Contexto dinámico Es un contexto cuyos elementos se definen o alteran durante la navegación. Se puede utilizar este tipo de contexto para la creación, modificación o exclusión de las instancias de una clase navegacional. Los historiales de navegación o las cestas de la compra son ejemplo de este tipo de contexto.

Para cada contexto de navegación se puede definir una estructura de acceso. Las estructuras de acceso son índices que permiten acceder al contexto. La especificación de la navegación entre los elementos del contexto define cómo será la navegación dentro del contexto, caracterizando la forma en la que se pueden recorrer los elementos de un contexto dado. Cuando un usuario entra en un contexto, el camino de navegación que puede seguir dependerá de la naturaleza del contexto y del tipo de especificación que se haga. Los tipos de navegación que OOHDM permite dentro de los contextos son los siguientes:

Navegación secuencial Una vez que el usuario entra en el contexto, puede navegar por todos los elementos del mismo siguiendo un orden secuencial y preestablecido. En este caso, se han de definir los conceptos primero, último, anterior y siguiente para el contexto.

Navegación circular Los elementos pueden verse como si estuvieran en una lista circular. Aquí solamente se definen los conceptos anterior y siguiente.

Navegación limitada al índice La navegación se realiza desde el índice a un elemento del contexto y viceversa. No hay posibilidad de navegar entre los otros elementos del contexto, a no ser que se vuelva al índice de entrada del contexto.

Combinación de navegación por índice y secuencial Se puede acceder a los elementos del contexto de forma indistinta a través del índice, o bien a través de los enlaces anterior y/o siguiente.

Navegación libre Se puede acceder en todo momento a todos los elementos del contexto de forma libre, de manera que no es necesario seguir ningún camino predeterminado para acceder a un elemento concreto.

5.3.4 Hera-S

Hera-S [187] es un método de los clasificados como orientado a datos que aparece como un refinamiento de Hera, añadiéndole a éste un uso más extensivo de las consultas y almacenamiento mediante Sesame (un *framework* de código abierto que trabaja con el lenguaje Resource Description Framework (RDF)). Hera-S separa el modelado del dominio, la navegación y el diseño de la presentación. Además, Hera-S también mantiene un modelo de contexto para modelar al usuario y la adaptación basada en el contexto. La figura 5.9 muestra un mapa conceptual con las relaciones entre los constructores usados en el modelo de aplicación de Hera-S.

El modelo de dominio describe la estructura de los datos y está basado en RDF. El propósito de este modelo es describir cómo el diseñador percibe la estructura semántica de los datos. Si se utiliza la herramienta que proporcionan los autores, los modelos se pueden definir en el lenguaje Ontology Web Language (OWL) [398]. Adicionalmente, los autores proponen o bien definir el modelo de dominio en UML y luego utilizar las conversiones de UML a OWL que se han publicado en la bibliografía [176], o bien utilizar directamente editores de ontologías como Protégé [124]. El modelo de contexto se implementa de forma similar al modelo de dominio, y su objetivo es capturar las relaciones entre la información relacionada con la personalización y adaptación del contenido. La principal diferencia entre ambos modelos está en los datos, mientras que en el modelo de dominio lo que se refleja es la información que se va a presentar al usuario, los datos del modelo de contexto suelen ser datos usados de forma interna para desarrollar la personalización y adaptación del contenido, como por ejemplo datos de la sesión.

Una vez que se tienen estos dos modelos, el diseñador debe definir el modelo de aplicación basado en el modelo de dominio. El modelo de aplicación describe la estructura de navegación y permite especificar cómo se estructura el acceso a datos describiendo qué datos se muestran al usuario y a qué páginas web puede navegar. A la vez, el modelo de aplicación permite que esta especificación sea dinámica personalizándola para

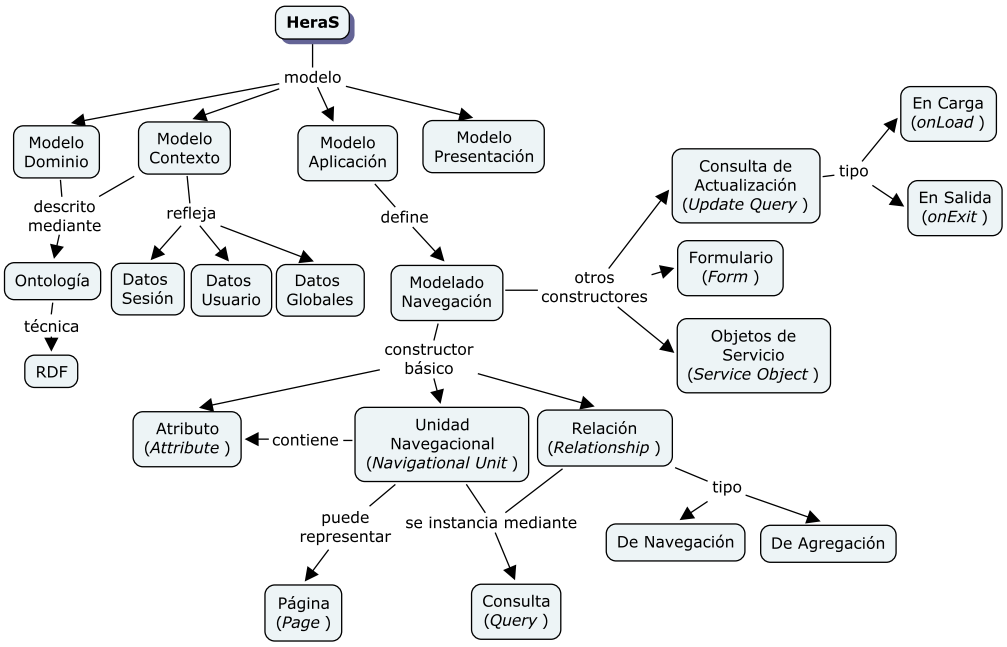


Figura 5.9: Esquema conceptual de los principales términos que intervienen en Hera-S

un usuario y adaptándola para un contexto específico. Aunque en las primeras versiones de Hera, los autores proporcionaban un editor y un lenguaje gráfico para definir el modelo de navegación, actualmente están dejando que esta funcionalidad recaiga sobre herramientas de terceros, que trabajan con las recomendaciones `RDF` y `OWL`. Para definir el modelo de aplicación, los autores usan Terse RDF Triple Language (`Turtle`) [34] y Sesame RDF Query Language (`SeRQL`) [46].

Finalmente, el diseñador deberá definir el modelo de presentación, que describe cómo se le mostrará cada unidad navegacional al usuario. Actualmente, los autores definen el modelo de presentación utilizando plantillas en Extensible Stylesheet Language Transformations (`XSLT`).

En el modelo de aplicación, la navegación se especifica mediante *unidades navegacionales* y *relaciones* entre esas unidades. La instanciación de las unidades y las relaciones se definen mediante expresiones de consulta que se refieren al contenido y al contexto. Las unidades pueden usarse para representar una página y pueden verse como una forma de agrupar los elementos que se le mostrarán al usuario. Los elementos que se muestran al usuario se denominan *atributos*, de forma que se puede decir que las unidades construyen estructuras jerárquicas de atributos. Los *atributos* son los trozos de información que se muestran al usuario. Esta información puede ser constante (es decir, que está predefinida y no cambia) o se puede obtener a partir del modelo de dominio. Las *relaciones* se usan para enlazar unas unidades con otras, para ir construyendo jerárquicamente las páginas. Hay dos tipos de relaciones: las llamadas *relaciones*

de agregación, que expresan cómo se componen las unidades, y las *relaciones de navegación* que expresan cómo navegar de una unidad a otra. Las unidades de navegación solamente tienen una relación, que se seguirá cuando el usuario haga click en alguno de los elementos visibles de la unidad.

Además de los constructores mencionados anteriormente, que se pueden considerar como básicos, éstos se han ido ampliando conforme esta propuesta ha ido evolucionando. Así, aparece el concepto de *subunidad*. Las *subunidades* se pueden utilizar para agrupar aquellos elementos que tienen un destino común. Las *subunidades* de una unidad pueden tener relaciones distintas a las de su padre. También aparecen nuevos tipos de unidades. Uno de ellos es la *unidad de formulario* que es un tipo de unidad de navegación que normalmente contiene elementos de entrada. Otro son las *unidades lógicas* que contienen elementos no visuales.

5.3.5 Object-Oriented Web Solutions (OOWS)

Object-Oriented Web Solutions (OOWS) [123] es una metodología de ingeniería web que nace como una extensión del método OO-Method [299], en la que los aspectos estáticos y dinámicos del sistema se definen mediante tres vistas complementarias: un modelo de la estructura estática, para el que se emplean diagramas de clases; un modelo dinámico, que se especifica mediante diagramas de estado y diagramas de secuencia; y, un modelo funcional, que representa mediante una especificación textual cómo afecta cada servicio al estado local de los objetos involucrados en su ejecución. La figura 5.10 muestra un mapa conceptual con los principales conceptos manejados en esta propuesta.

Para tratar con los rasgos característicos de las aplicaciones web, OOWS introduce tres nuevos modelos: el modelo de usuario, el modelo de navegación y el modelo de presentación. En el modelo de usuario se especifican los tipos de usuario que pueden interactuar con el sistema. El modelado de la navegación se realiza desde dos perspectivas distintas: una especificación de los aspectos estructurales y globales de la navegación, lo que los autores denominan *Authoring-in-the-Large*, y otra, que se refiere al contenido de los nodos, llamada *Authoring-in-the-Small*.

En la fase de "autoría de lo grande" se describe la navegación permitida a cada uno de los tipos de usuario de la aplicación mediante *mapas de navegación*. Un *mapa de navegación* es un grafo cuyos nodos son o bien *contextos navegacionales* o bien *subsistemas de navegación*. Los primeros son unidades de interacción con el usuario que proporcionan un conjunto de datos cohesionados y un conjunto de operaciones, mientras que los segundos proporcionan una forma de estructurar los mapas de navegación cuando éstos son muy complejos. Los arcos del grafo representan *enlaces de navegación* que definen las rutas válidas en el sistema. En la fase de "autoría de lo pequeño" se especifican los contextos navegacionales con más detalles. Los contextos están formados por *Unidades de Información Abstractas* UIA, que representan los requisitos para recuperar un trozo de información. Hay dos tipos de UIA's, contextuales y no contextuales. Mientras que los primeros recuperan la información filtrándola mediante la información que transportan

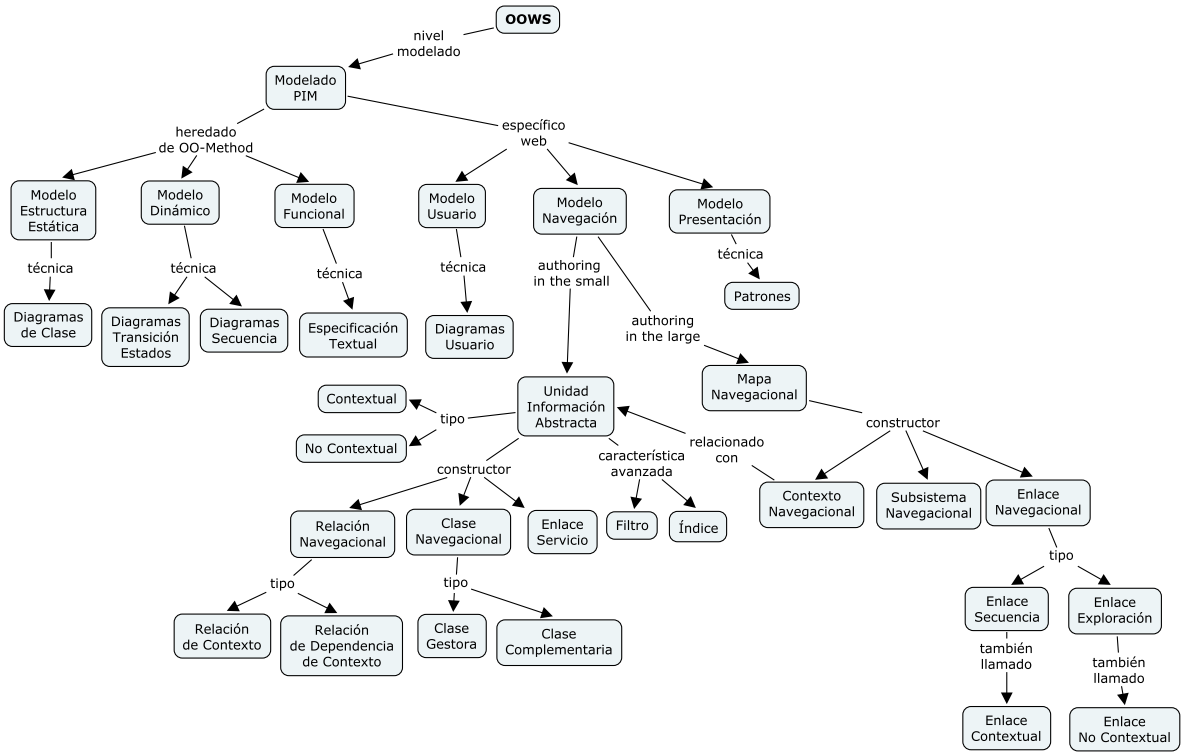


Figura 5.10: Mapa conceptual con la terminología usada en la propuesta OOWS

los enlaces de secuencia, los segundos no dependen de ningún enlace contextual.

Las UIA's están compuestas de *clases navegacionales* que son vistas de las clases conceptuales, y contienen la información que se le mostrará al usuario. Hay dos tipos de clases navegacionales: *clases gestoras* y *clases complementarias*. En cada UIA tiene que haber una clase gestora, pudiendo existir un número variable de clases complementarias. Las clases navegacionales se relacionan mediante *relaciones navegacionales*, que se definen sobre las relaciones del modelo de clases conceptual. En OOWS se definen dos tipos de relaciones: relaciones de dependencia del contexto y relaciones de contexto. La primera representa una relación en la que se recuperan todos los objetos relacionados con la clase original, la segunda representa lo mismo que la primera, pero con una capacidad de navegación a un contexto navegacional objetivo, creando un enlace de secuencia en el mapa navegacional.

Finalmente, el modelo de presentación es altamente dependiente del modelo navegacional. Se basa en los contextos de navegación para definir las propiedades de presentación. Los requisitos de presentación se especifican mediante una serie de patrones que se asocian a las primitivas del contexto navegacional.

5.3.6 WebDSL

WebDSL [392, 405] es un Lenguaje Específico de Dominio(DSL) para implementar aplicaciones web dinámicas con un modelo de datos rico. Inicialmente era un proyecto que comenzó como un ejercicio para trabajar con el diseño e implementación de DSLs, aunque se ha convertido en una herramienta para construir aplicaciones web en producción. En WebDSL se proporcionan varios sublenguajes relacionados con los *concerns* necesarios para construir una aplicación web, tales como modelos de dato, plantillas de interfaz de usuario, reglas de control de acceso, reglas de validación de datos y *workflows*.

Las interfaces en WebDSL se definen como una combinación de llamadas a plantillas, primitivas y secuencias de escape HTML. WebDSL manipula internamente esta combinación de elementos y genera código para cada uno de ellos.

La navegación en WebDSL no se trata de forma separada, sino que está embebida en el sublenguaje de interfaz de usuario en forma de primitivas. En concreto, para implementar la navegación se utiliza la primitiva `navigate`. La llamada a `navigate` crea un enlace a la página que se le pasa como argumento. La figura 5.11, tomada de [152], muestra un ejemplo del uso de esta primitiva. En la parte superior se puede ver cómo se hace una llamada a `navigate`, mientras que en la parte inferior se muestra la definición de la página destino del enlace.

Figura 5.11: Ejemplo del constructor navigate en WebDSL

5.3.7 Otras propuestas que modelan la navegación desde una perspectiva del comportamiento

La especificación del comportamiento de la navegación se centra en la interacción del usuario con la aplicación, fijándose en qué ocurre cuando el usuario navega o invoca una funcionalidad proporcionada por la navegación web. Los formalismos para modelar el comportamiento normalmente se basan a principalmente en modelos de transición de estados, y en sus equivalentes, tales como las redes de Petri, y normalmente se utilizan en propuestas que siguen un enfoque ascendente.

Muchos investigadores se han decantado por los diagramas de estados (*statecharts*) como notación de modelado para la hipermedia y las aplicaciones web [115, 411, 231, 148, 88, 82] y las interfaces de usuario [186]. Lo que se modela en este tipo de propuestas son los distintos caminos en un espacio de navegación que está formado por transiciones y estados. Normalmente un estado representa un trozo de información observable por el usuario en un nodo de hipertexto en un momento dado. Una transición permite moverse de un estado a otro y normalmente se dispara por un evento relacionado con la interacción del usuario o con otro tipo de eventos temporales o generados automáticamente por el sistema.

HBMS Hyperdocument Model Based on Statecharts (`HBMS`) [115] es un modelo orientado a la navegación y basado en *statechart* para la especificación de hiperdocumentos. Los modelos `HBMS` usan una simplificación de la notación de máquinas de estados, en la que los estados representan páginas y las transiciones representan enlaces de navegación entre estados. `HBMS` no está pensado para aplicaciones web en sí, sino para hiperdocumentos. Como consecuencia de esto, la notación no incluye acciones. Tampoco contempla la noción de servidor o de aplicación que tiene que ejecutarse, ya que su objetivo es especificar un hiperdocumento que puede verse o navegarse. Esto implica que esta propuesta de modelado es buena para modelar páginas web estáticas con los enlaces definidos como anclas.

Dolog Dolog [88] utiliza los diagramas de estado para modelar la navegación adaptable. El modelado de la navegación se basa en el *browsing*, así que se concentra exclusivamente en la interacción del usuario durante la presentación del hipertexto o en un cambio de estado cuando el usuario navega. De esta forma, se excluyen las acciones en el servidor. Esta propuesta también se basa de forma estricta en `UML`, y aunque para llegar a obtener el diagrama de estados hay una serie de pasos previos, en este apartado nos vamos a centrar exclusivamente en el modelado mediante diagramas de estado. Los estados aquí representan trozos de información. Se usan condiciones en las transiciones para indicar los nodos que son accesibles para un determinado tipo de usuario, es decir, representan características variables tanto en la información como en el destino de los enlaces.

5.4 Propuestas orientadas al análisis, verificación y pruebas de sitios web

El principal objetivo de las propuestas de este grupo es ser utilizados en el análisis, la verificación y pruebas de sitios web, y se pueden clasificar de acuerdo a las características capturadas por los modelos y las propiedades que las mismas son capaces de comprobar en las siguientes categorías [6]: métodos que se centran en el modelado de la interacción del usuario con el navegador, métodos que se centran en el modelado del contenido, métodos que se centran en el modelado de la navegación y métodos híbridos, que cubren más de una de estas características.

5.4.1 Formal Approach for Rich Navigation (FARNav)

Formal Approach for Rich Navigation (`FARNav`) [170] es una propuesta que se centra en la separación del código de enrutamiento de la navegación en aplicaciones `J2EE`, en el modelado de este enrutamiento y en su relación con el modelo de navegación. El código de enrutamiento es el código relacionado con el encaminamiento de una petición de una página web a través de los diferentes componentes de un servidor. Se puede clasificar como propuesta que se centra en el modelado de la navegación. Los autores de esta propuesta definen una notación propia para modelar el enrutamiento de las peticiones, dan unas guías para implementar el enrutamiento utilizando AspectJ [309] y definen las relaciones existentes entre el modelo de navegación y el modelo de enrutamiento. Mientras que el modelo de navegación describe las posibles secuencias de páginas web que un usuario puede visitar, el modelo de enrutamiento describe cómo los componentes del servidor manejan cada petición. A modo de resumen, en la figura 5.12 se muestra un mapa conceptual con los principales términos usados en esta propuesta y la relación entre los mismos.

`FARNav` usa *statecharts* para describir de forma precisa la navegación. Para tratar con la navegación adaptativa, es decir, aquella navegación en la que el siguiente paso a dar depende de cosas tales como el tipo de usuario o las páginas que el usuario ha visitado antes, se utilizan *statecharts* con estados paralelos. El subestado principal contiene una máquina de estado que tiene un estado por cada página web y transiciones entre páginas para los enlaces de navegación. Si la aplicación no tiene ningún tipo de navegación adaptativa, éste será el único subestado que habrá. Si la aplicación presenta navegación adaptativa, entonces se define un subestado paralelo por cada uno de los modos en los que esté el usuario. Cada uno de estos subestados tendrá su propia máquina de estados. Estos subestados son paralelos porque un usuario puede estar en más de un modo a la vez. En esta propuesta solamente se modela la navegación entre páginas, no se modela ni la navegación dentro de una página ni los marcos de forma explícita.

La máquina de estados para la navegación de páginas se construye de la siguiente forma: cada página web es un estado separado. Cuando un usuario puede llegar a una página desde otra, se define una transición entre sus estados. La transición llevará

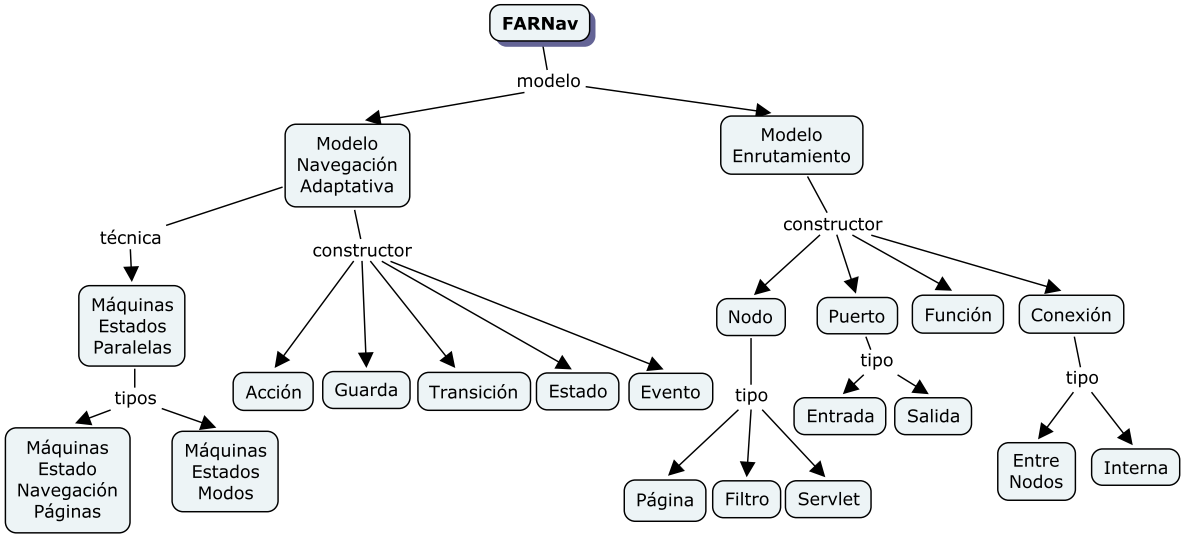


Figura 5.12: Esquema conceptual de los principales términos que intervienen en FARNav.

una etiqueta asociada con la forma `<evento> [<guarda>] / <acción>`. El `evento` es obligatorio y describe una acción del usuario que provoca que se envíe una petición al servidor. La `guarda` es opcional y permite que una transición se dispare solamente cuando un modo de usuario tiene un valor particular. La transición dispara una `acción` que aparece como un evento en una máquina de estado de un modo de usuario.

La máquina de estados para los modos se crea así: se define un estado por cada uno de los posibles valores del modo. La navegación basada en la historia de navegación también se modela como un modo, de tal forma que hay un estado para indicar el punto en las páginas visitadas con anterioridad que no son relevantes, y un estado por cada caso en el que se debe recordar una de las páginas visitadas. Las transiciones se disparan normalmente con eventos que tienen lugar en la máquina de estados para la navegación de páginas, o con eventos temporales tal como cuando transcurre un cierto periodo de tiempo. En estas transiciones no hay guarda ni acción.

Por otra parte, el modelo de enrutamiento está formado por una serie de nodos que representan componentes del servidor, pudiendo ser páginas, filtros o *servlets*. Cada nodo tiene unos puertos de entrada y de salida asociados. Cada puerto tiene asociado un identificador de recursos universal (`URI`). Un nodo con un puerto de entrada indica que ese nodo recibe peticiones con la `URI` asociada a ese puerto, mientras que un nodo con un puerto de salida envía peticiones a la `URI` asociada con ese puerto. El puerto de entrada de un nodo se conecta con el puerto de salida de otro nodo cuando sus `URI`'s son las mismas. Un nodo puede tener conectado algún puerto de entrada con algún puerto de salida. Esta conexión dentro del propio nodo puede tener asociada una función que procese las peticiones que lleguen con el `URI` asociado al puerto de entrada. Una función

se puede asociar con varias conexiones dentro de un nodo.

Uno de los objetivos de esta propuesta es la utilización de estos modelos para verificar los sitios web. Para conseguirlo, los autores definen una serie de transformaciones de los modelos de navegación a Computational Tree Logic (CTL) [103], un lenguaje que sirve como entrada a Symbolic View Modifier (SVM) [242], una herramienta ya existente para chequeo de modelos.

5.4.2 Web Applications Analysis and Testing (WAAT)

Web Applications Analysis and Testing (WAAT) es una propuesta de ingeniería inversa que se usa para extraer modelos UML con una interacción mínima por parte del usuario y se centra en el estudio de aplicaciones web heredadas en las que la lógica de negocio está embebida en las páginas web. Se usan diagramas de clases para describir la estructura y los componentes de la aplicación web, mientras que se usan diagramas de estado para representar el comportamiento y las estructuras de navegación. De esta forma, se puede clasificar como una propuesta híbrida. Como resumen, en la figura 5.13 se muestra un mapa conceptual con los principales términos manejados en esta propuesta.

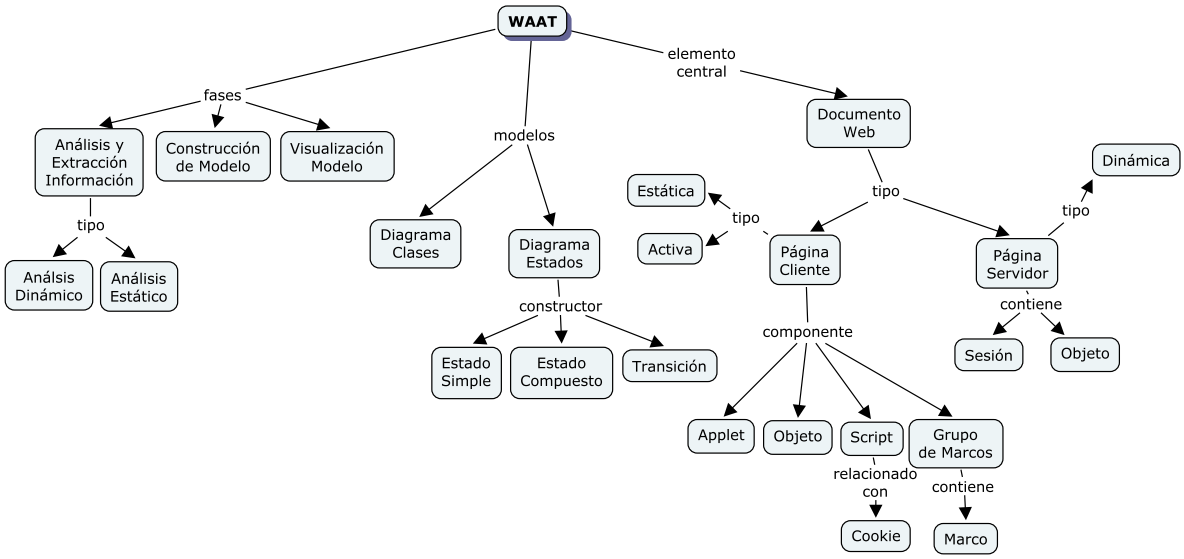


Figura 5.13: Esquema conceptual de los principales términos que intervienen en WAAT.

Una aplicación web está formada por varios componentes, que se pueden clasificar en: documentos web (o páginas), objetos web (o componentes compilados que proporcionan algún servicio al resto de la aplicación a través de una interfaz bien definida), y objetos y componentes del servidor (bases de datos, sistemas heredados, componentes del servidor). Esta propuesta se centra en los documentos web que los autores clasifican como estáticos, activos o dinámicos. Un *documento estático* es una página simple

situada en el servidor, cuyo contenido es fijo y solamente se modifica cuando se edita el fichero en el que está definida. Un *documento activo* es aquél que llega a la aplicación cliente y que necesita alguna interpretación por parte de ésta antes de ser mostrado (normalmente suelen ser páginas con código de *script*). Finalmente, un *documento dinámico* es aquél que se genera en el servidor por alguna petición de un cliente web, es decir, en el servidor se ejecuta algún programa que permite generar la salida que se enviará al cliente.

WAAT comprende varias fases: el análisis y la extracción de información; la construcción del modelo, y, finalmente, la visualización del modelo. La fase de análisis y extracción se basa en el análisis del código fuente. El enfoque tradicional para analizar las páginas web dinámicas consiste en aplicar un análisis estático a cada una de las páginas que se pueden generar en el servidor. Los autores de esta propuesta realizan el análisis de las páginas web mediante dos técnicas: un análisis dinámico para definir los caminos de ejecución relevantes, y un análisis estático para analizar los caminos definidos durante el análisis dinámico. De esta forma, no se analizan todas las posibles páginas generadas, sino solamente aquéllas relevantes. El análisis dinámico se basa en el análisis de mutación y la simulación de sesiones de navegación, mientras que el análisis estático se basa en analizadores léxico/sintácticos (*parsers*). En WAAT se obtienen dos tipos de modelos: diagramas de clases, para describir la estructura y los componentes de la aplicación web; y diagramas de estado, que representan el comportamiento y la estructura de navegación.

La estructura navegacional está formada por páginas cliente/servidor, enlaces de navegación, conjuntos de marcos, formularios de entrada, flujo de control del código de los *script*, y otro contenido estático y dinámico. El diagrama de estados se utiliza para modelar algunos recursos, tales como un documento activo. El diagrama de estados de un documento activo puede definir el flujo de llamadas a funciones del código de *script*, y alguna información dinámica de comportamiento o navegación, como enlaces dinámicos, marcos y demás.

Con respecto a los diagramas de estados, los documentos estáticos se representan mediante un estado simple. Los documentos activos se representa por un estado compuesto, que puede ser concurrente si la página contiene código de *scripts* en la parte del cliente. Los documentos dinámicos se pueden modelar tanto con estados simples como con estados compuestos. Si el documento no contiene ningún elemento relevante de navegación, entonces se modela mediante un estado simple, si no, se modela como un estado compuesto. Por ejemplo, una página dinámica que genera varios documentos HTML en la parte del cliente, se modela como un estado compuesto por varios estados simples, uno por cada una de las páginas que se genera.

5.4.3 Otras propuestas de análisis y verificación que usan *statecharts*

Lo que se modela en este tipo de propuestas son los distintos caminos en un espacio de navegación que está formado por transiciones y estados. Normalmente un estado

representa un trozo de información observable por el usuario en un nodo de hipertexto en un momento dado. Una transición permite moverse de un estado a otro y normalmente se dispara por un evento relacionado con la interacción del usuario o con otro tipo de eventos temporales o generados automáticamente por el sistema.

Algunas de las propuestas que usan *statecharts* para modelar la navegación son:

StateWebChart StateWebChart (SWC) [411] es un formalismo basado en *statecharts* para especificar la navegación de las aplicaciones web. SWC se presenta como un modelo de navegación abstracto sin ninguna referencia a cómo se ejecutará el modelo.

Como SWC es una extensión a los *statecharts* tradicionales, se introducen nuevos tipos de estado (básico, transitorio, dinámico y externo) y nuevos tipos de transiciones (de usuario, del sistema, de terminación), dependiendo de quién produce el evento que dispara la transición.

Leung et al. Leung et al. [231] proponen el uso de los *statecharts* para especificar el modelo navegacional de las aplicaciones web. En esta propuesta se intenta modelar la navegación de forma independiente a la tecnología de implementación. Los estados representan páginas que se verán en un navegador, y los estados concurrentes representan múltiples páginas o partes de una página que se muestran de forma concurrente en el navegador. Las transiciones representan enlaces y no se tiene en cuenta el disparo de acciones en el servidor.

Gorshkova y Novikov En [148] lo que se hace es extender la semántica de los *statecharts* mediante la definición de diferentes tipos de estados (página), todos ellos relacionados con lo que se puede ver en un navegador, y tipos de transiciones (de cliente, y del sistema). Esta propuesta se basa estrictamente en `UML` y las extensiones se definen mediante estereotipos. Las acciones se excluyen de forma explícita, porque el objetivo es el modelado de la navegación y no del comportamiento dinámico.

5.5 Sumario

En este capítulo se ha dado una visión general de cómo las propuestas de ingeniería web se enfrentan al modelado de la navegación. Como se ha podido comprobar hay un número muy elevado de propuestas con una terminología, aunque bastante cercana, heterogénea. En [330] se hacía una revisión de las distintas propuestas de diseño de aplicaciones web existentes hasta el momento, con objeto de ver si cubrían una serie de requisitos, como por ejemplo si se enfrentaban a la ubicuidad o la adaptación. Una de las conclusiones que ya se obtenía en este trabajo era que existían demasiadas propuestas con demasiadas notaciones distintas, resaltando la necesidad de la definición de un lenguaje estándar con una notación comúnmente aceptada tanto desde el mundo de la empresa como desde la academia. Hoy en día se están haciendo esfuerzos por acercar

posturas dentro del proyecto MDWEnet [387], que aboga por conseguir que las diferentes propuestas de ingeniería web puedan interoperar.

Con respecto a la navegación, en [322, 323] se hacía una primera aproximación al estudio de la navegación y se proponía tratarla como un aspecto separado, mientras que en [319] se muestra una comparativa de las propuestas de ingeniería web que se han tratado en este capítulos desde varias perspectivas, entre las que se encuentra la navegación.

The increment of meaning corresponds to the increased perception of connections and continuities of the activities in which we are engaged.

John Dewey

6

Desarrollo de Software Orientado a Aspectos y Dirigido por Modelos en Entornos Web

El objetivo de este capítulo es presentar cómo algunas de las propuestas de ingeniería web están adoptando un enfoque orientado a aspectos y dirigido por modelos. Para ello se estructura de la siguiente forma: en primer lugar, en la sección 6.1 se introduce el capítulo. Luego, en la sección 6.2, se muestra cómo las propuestas de ingeniería web han adoptado algunas de las ideas propuestas en el área de la orientación a aspectos. La sección 6.3 se centra en ver la adopción de las ideas del desarrollo de software dirigido por modelos en la comunidad de desarrollo web. Por último, se resume el capítulo.

6.1 Introducción

Algunos autores que trabajan en ingeniería web se están fijando en las ideas que subyacen en la orientación a aspectos y en el desarrollo de software dirigido por modelos [319]. Tradicionalmente, las propuestas de modelado de aplicaciones web han usado vistas para separar los modelos conceptuales, de navegación y de presentación. Pero algunas de las propuestas de ingeniería web se han percatado de que existen otros conceptos, como la adaptación o la personalización, que se diseminan y atraviesan a estos modelos, y que que se modularizan mejor si se tratan como conceptos en el sentido de la orientación a aspectos. En este capítulo se pretende mostrar cómo éstas ideas están calando en la ingeniería web.

De la misma forma, aunque ya existía una gran tradición en el uso de modelos dentro

de la ingeniería web, muchas de las propuestas están siendo revisadas y están incorporando las nuevas tendencias que han aparecido en el área del desarrollo de software dirigido por modelos. Como consecuencia de esto, muchas de las propuestas que utilizaban la manipulación directa de modelos como base de sus herramientas CASE, están adoptando el uso de los nuevos lenguajes de transformaciones que han surgido en el área. De igual manera, la propuesta MDA también ha sido bien acogida en la comunidad web.

6.2 Orientación a Aspectos en la Ingeniería Web

La rápida evolución de las aplicaciones web debido al uso de nuevas tecnologías, lenguajes y modelos de programación, provoca que las metodologías de análisis, diseño, modelado y verificación de sitios web estén en constante proceso de revisión y evolución, tal y como lo demuestran el gran número de trabajos comparativos y evaluaciones de las distintas propuestas existentes en la bibliografía [319]. Una de las evoluciones más recientes se debe a que los autores de las propuestas de modelado de aplicaciones web se están fijando en las ideas que subyacen en el Desarrollo de Software Orientado a Aspectos y las están incorporando a sus metodologías. En este apartado se describe cómo en el campo de la ingeniería web se están adoptando estas ideas. Para ello se indica cómo las propuestas descritas en el capítulo anterior han acogido estos conceptos.

6.2.1 aspectWebML

aspectWebML [347] define una extensión orientada a aspectos para WebML. El objetivo principal de esta propuesta es modelar la personalización en las aplicaciones web ubicuas, considerando como tales aquellas aplicaciones web a las que se accede desde diferentes entornos, es decir, tanto desde cualquier dispositivo móvil, como por usuarios con diferentes intereses y, por último, en cualquier momento y lugar. Los autores de esta propuesta conciben la personalización como un concepto que se esparce por los diferentes niveles (contenido, hipertexto y presentación) de una aplicación web y, por lo tanto, es susceptible de tener el tratamiento de lo que en programación orientada a aspectos se conoce como *crosscutting concern*. De esta forma, los autores definen una extensión a WebML basada en un marco conceptual de referencia para el modelado orientado a aspectos.

El marco conceptual de referencia ha sido diseñado para abstraerse de los diferentes mecanismos de composición que existen en las propuestas de modelado orientadas a aspectos, de tal forma que se pueden definir especializaciones bien para utilizar el mecanismo asimétrico (punto de corte-*advice*), bien para utilizar un mecanismo de composición simétrico (cf. sección3.2). Aunque el marco conceptual permite diferentes mecanismos de composición, en la extensión de WebML los autores se han decantado por un enfoque asimétrico, argumentando que este enfoque es más factible que el simétrico porque la personalización representa un concepto que no existe por sí mismo, sino que

tiene que aplicarse en relación a la funcionalidad de la aplicación web. De esta forma, en la propuesta se cuenta con un modelo base descrito mediante los modelos de contenido e hipertexto de `WebML`, mientras que la personalización se define mediante el lenguaje `aspectWebML`.

En la figura 6.1 se muestra un diagrama de clases de los elementos con los que se ha extendido `WebML` y de sus relaciones. La extensión se consigue haciendo que la metaclase `ModelElement` extienda a las metaclases `JoinPoint` y `ComposableElement`, indicando de esta manera que un elemento de modelado definido en `WebML` puede servir como un punto de enlace (desde el punto de vista de un mecanismo de composición asimétrico) o un elemento a componer (desde un punto de vista de composición simétrico), aunque como se ha comentado anteriormente, este segundo caso se ha pospuesto como trabajo futuro. Además de estas dos metaclases, se añaden `SimpleAdvice` y `ConcernModule`.

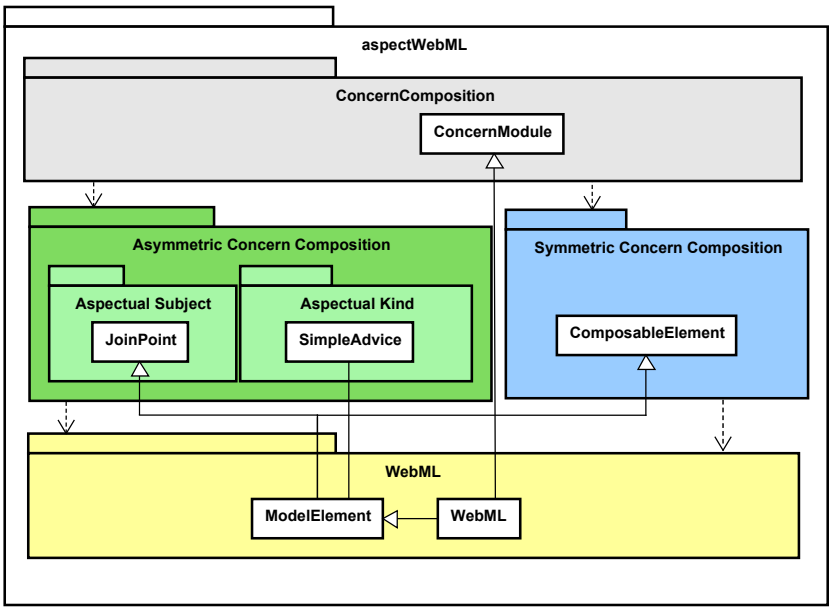


Figura 6.1: Extensiones realizadas a WebML para incorporar una filosofía orientada a aspectos.

Finalmente, se muestra un extracto, tomado de [347], de una aplicación que utiliza `aspectWebML` para definir una extensión que consiste en proporcionarle al usuario información de noticias y eventos. En la figura 6.2, se muestra una regla de composición asimétrica con la que se pretende añadir páginas de tipo *Landmark* a la vista del sitio web que está disponible para todos los usuarios. Así, se define un *advice* en el que se incluyen las páginas `News` y `Events` como hitos para las noticias y los eventos, respectivamente. De igual manera, se define un punto de enlace `PublicSiteView` que indica que los hitos definidos en el *advice* se añadirán a la vista del sitio `Public`.

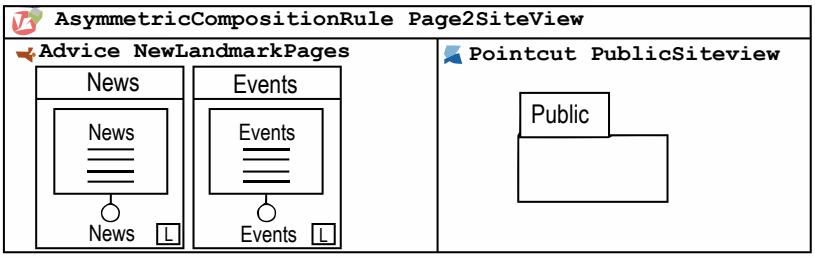


Figura 6.2: Ejemplo de modelado de personalización en aspectWebML

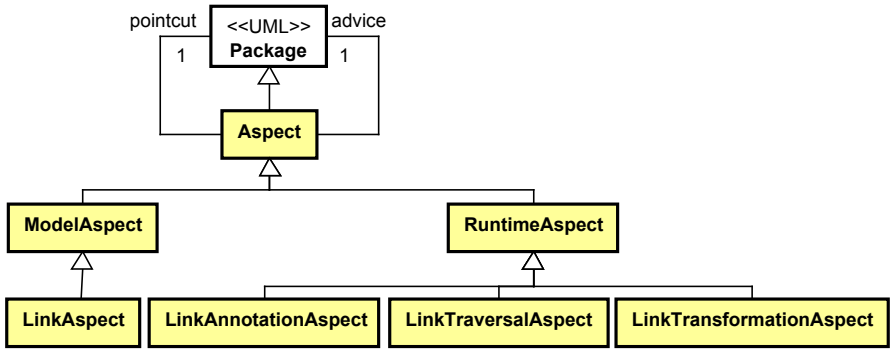
6.2.2 Extensión Orientada a Aspectos de UWE

Los autores de `UWE` también han incorporado ideas de orientación a aspectos para tratar con aplicaciones web adaptativas [219], que son aquéllas en las que el contenido que se muestra al usuario depende del usuario concreto que esté interactuando con la aplicación. Al igual que ocurría con la propuesta anterior, la adaptabilidad es considerada como un concepto ortogonal a las vistas de contenido, navegación y presentación, de forma que para modelarla de forma no invasiva, se usan técnicas de modelado orientadas a aspectos y un enfoque asimétrico.

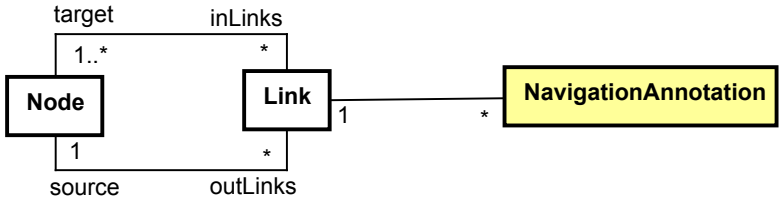
Para incorporar las ideas de la orientación a aspectos los autores definen, por una parte, una extensión ligera a `UML`, y por otra, una extensión a su propia metodología [31]. En la extensión a `UML` (figura 6.3(a)) los autores definen los conceptos de aspecto, punto de enlace (*pointcut*) y *advice* como subclases de la metaclase `Package` de `UML`. De tal forma que un aspecto puede verse como un paquete que contiene exactamente un `advice` y un `pointcut`, que también se ven como paquetes. La semántica de aplicación de los `advices` en los `pointcut` depende del tipo de aspecto definido. Los autores han definido dos tipos de aspectos: aspectos de modelado (`ModelAspect`) y aspectos en tiempo de ejecución (`RuntimeAspect`). Los primeros se tejen de forma estática a nivel de modelos, mientras que el efecto resultado de tejer los segundos se ve en tiempo de ejecución, ya que para hacer el tejido se necesita información que solamente está disponible en tiempo de ejecución. Como se puede comprobar en la figura 6.3(a), de los posibles niveles de adaptación [217] (de contenido, de navegación y de presentación), solamente se afronta la adaptación de los enlaces, limitándose a cuatro técnicas: *ordenación de enlaces adaptativa*, que ordena un conjunto de enlaces de un nodo particular; *anotación de enlaces adaptativa*, que consiste en añadirle a los enlaces información textual o iconos, proporcionando información adicional al usuario; *ocultación de enlaces adaptativa*, que consiste en borrar, ocultar o deshabilitar una serie de enlaces; y, *generación de enlaces adaptativa*, que es una característica en tiempo de ejecución que permite añadir nuevos orígenes o destinos a los enlaces de un nodo basados en el estado actual del usuario.

En la extensión a `UWE` (figura 6.3(b)) se añade una nueva metaclase llamada `NavigationAnnotation`, que se relaciona con los enlaces de navegación definidos en `UWE`

(metaclase `Link`), y se utiliza para capturar la ordenación de enlaces adaptativos, la anotación de enlaces, y la ocultación de enlaces.



(a) Extensión al metamodelo de UML



(b) Extensión al metamodelo de UWE

Figura 6.3: Extensiones realizadas en UWE para tratar con aspectos

La extensión a `UWE` solamente se aplica al control de acceso y a la adaptación de la navegación, y todos los puntos de corte se definen sobre el modelo de navegación. En la figura 6.4, se muestra un ejemplo de aspecto en `UWE`. Con este aspecto se define un punto de enlace, de tal forma que se seleccionan todas las clases del modelo de navegación que se llamen `RegisteredUser`, y mediante el *advice* se le añade un nuevo método denominado `visited`.

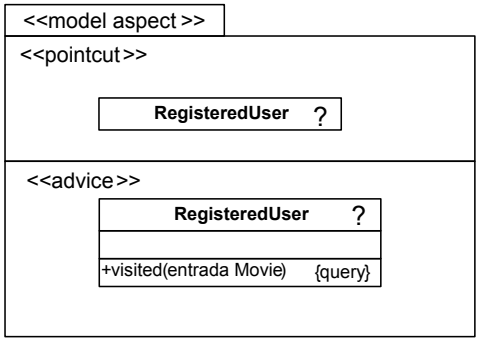


Figura 6.4: Ejemplo de aspecto de modelado definido en UWE.

6.2.3 Separación Avanzada de Conceptos en OOHDM

Esta propuesta puede ser considerada como una evolución de OOHDM para tratar con los *crosscutting concerns*, y está inspirada por [257], por lo que siguen una aproximación simétrica. Los autores apuestan por una captura de los *crosscutting concerns* en etapas tempranas de desarrollo, de tal forma que, al igual que en [256], consideran un *concern de una aplicación* como un conjunto de requisitos que se refieren a un mismo tema o a una característica del comportamiento de una aplicación. Así, se ha de comenzar por una identificación de los *concerns* presentes en el dominio del problema. Para ayudar en esta identificación los autores proponen el uso de un catálogo de *concerns*. Además, éstos se representan utilizando una serie de plantillas basadas en XML y se construye un diagrama de interacción con el usuario (User Interaction Diagram (UID)). El diagrama de interacción con el usuario es una técnica que permite especificar mediante diagramas de estado los elementos que se le presentarán al usuario y las transiciones correspondientes a las acciones de los mismos.

Una vez recolectados y modelados todos los requisitos, se construye un modelo conceptual de OOHDM, basándose en la información obtenida en los diagramas de interacción con el usuario. El modelo conceptual se divide siguiendo el enfoque Theme [60], de tal forma que se define un submodelo conceptual por cada uno de los conceptos relevantes. El siguiente paso es definir el modelo navegacional, tal y como se ha explicado en la sección correspondiente a la metodología OOHDM, y determinar cuáles son los nodos afectados por los *concerns* existentes. Para recoger esta información, los autores usan una tabla en la que por filas se colocan los diferentes *concerns* de la aplicación, y por columnas, los nodos del modelo de navegación. En cada casilla se indica de qué forma afecta el *concern* al nodo. A continuación, y basados en el concepto de rol, los autores definen un nuevo diagrama de navegación llamado diagrama de navegación sensible al *concern*. Estos diagramas son iguales que los diagramas de navegación de OOHDM, pero a cada nodo se le añade como decorador un rol por cada uno de los *concerns* que le afectan. En el rol se incluye la información con la que el *concern* concreto enriquece al nodo. La figura 6.5, tomada de [197], muestra un diagrama de navegación sensible al contexto para una aplicación de comercio electrónico tipo *Amazon*. En ella se puede ver cómo el nodo *Producto* (`Product`) está afectado por los *concerns Recomendar Producto* (`Recommend Product`) y *Producto en Carrito de la Compra* (`Product In Cart`). En el primer caso, al nodo *Producto* se le añade información para explicar el motivo de la recomendación (atributo `Why`) y para navegar, o bien al siguiente producto recomendado, o bien al anterior (mediante los atributos `Next` y `Prev`, respectivamente). En el segundo caso, el nodo se enriquece con dos atributos (`Message` y `OtherProducts`) y un método (`AddToCart`).

Los autores de esta propuesta se han centrado en estudiar los *concerns* navegacionales (aquéllos que se manifiestan en la estructura de navegación de la aplicación) [146, 197], pero también han publicado trabajos estudiando los *concerns* que aparecen en aplicaciones de hipermedia física (aquellas aplicaciones que tratan con

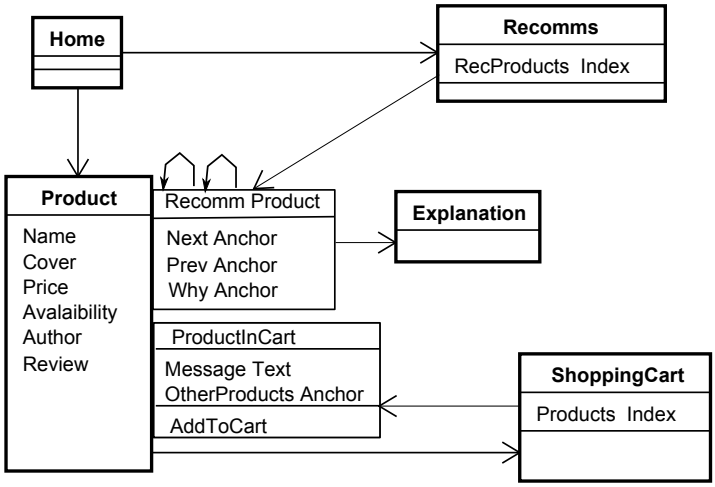


Figura 6.5: Diagrama de navegación sensible al concepto

realidad aumentada) [147] y en la interfaz de usuario [142]. Finalmente, aunque las principales contribuciones de esta propuesta son a nivel de diseño, los autores también se han enfrentado a aspectos de implementación para tratar con la funcionalidad volátil [343], es decir, aquélla que por su propia naturaleza sólo está vigente durante un periodo de tiempo determinado. Un ejemplo de esta funcionalidad sería el poner a la venta entradas para los conciertos de la gira de un grupo en una aplicación de comercio electrónico, ya que esta funcionalidad solamente estará disponible mientras dure la gira del grupo.

6.2.4 Semantic-based Aspect-Oriented Adaptation Language (SEAL)

Semantic-based Aspect-Oriented Adaptation Language (`SEAL`) [56] es un lenguaje específico de dominio que se ha definido para introducir la adaptación en el contexto de Hera-S. El objetivo de los autores de esta propuesta era extender las aplicaciones web existentes con nueva funcionalidad (de adaptación) sin rediseñarlas o reimplementarlas completamente. Para cumplir este objetivo, y basándose en la naturaleza de *crosscutting concern* de la adaptación, se ha seguido un enfoque asimétrico (punto de corte-*advice*). En la figura 6.6, se muestra un extracto de un modelo de aplicación de Hera-S, definido con el lenguaje `Turtle`, donde aparece una Unidad de Navegación que hace referencia a un director de una película. Recuadrado se encuentra el código `SeRQL` correspondiente a una adaptación al contexto, en el que se indica que la foto del director solamente se mostrará si no se accede a la unidad desde una `PDA`. Es muy probable que este trozo de código remarcado aparezca en más lugares del modelo de aplicación, ya que habrá más atributos de tipo fotografía en la aplicación (por ejemplo, los actores) que habrá que ocultar por cuestiones de eficiencia si se accede a las páginas que las contienen mediante una `PDA`.

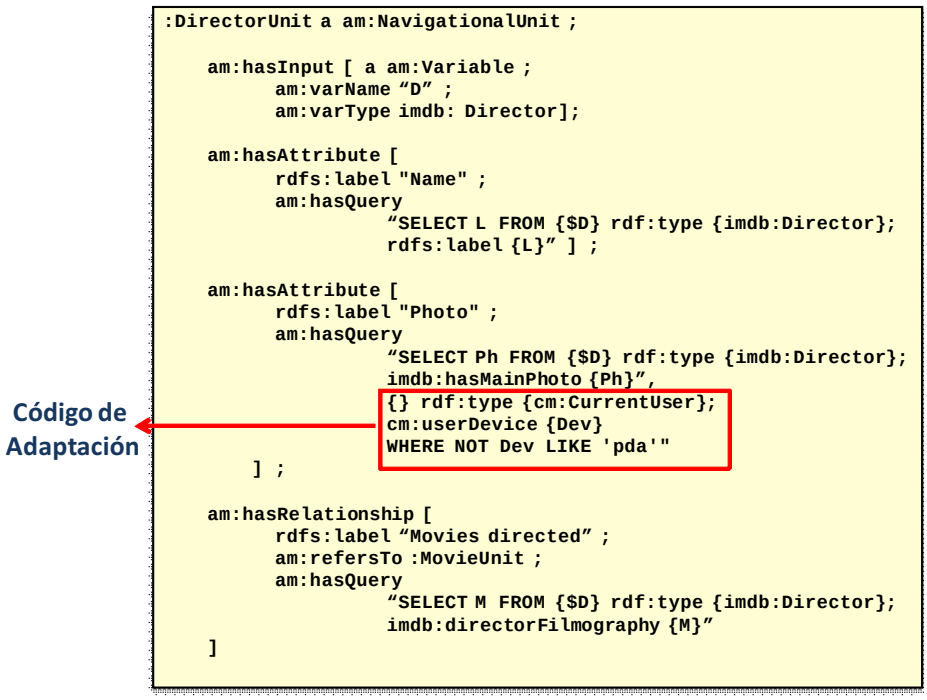


Figura 6.6: Extracto de un modelo de aplicación en Hera-S con una adaptación al contexto

Así, en este contexto, los puntos de corte se definen como consultas sobre el Modelo de Aplicación, determinando así qué elementos requieren una adaptación. El *advice* comprenderá un conjunto de acciones de adaptación. Estas acciones servirán para inyectar condiciones de adaptación en los puntos de corte definidos, y estarán formadas por un conjunto de transformaciones arbitrarias aplicadas a los elementos del modelo de aplicación que hayan sido seleccionados por los puntos de corte. Para definir estos puntos de corte y *advices* los autores han definido su propio lenguaje orientado a aspectos, `SEAL`.

La definición de puntos de corte en `SEAL` se basa en ir restringiendo los elementos del modelo de aplicación, de tal forma que se pueden especificar expresiones para restringir el tipo del elemento al que se va a aplicar la adaptación, definir condiciones que deben cumplir los elementos, o incluso definir llamadas nativas al lenguaje de consultas `SeRQL` para definir expresiones más complejas.

Los *advices* se definen mediante tres tipos de operaciones básicas de adaptación: añadir condiciones a los elementos, añadir o borrar elementos, y, finalmente, reemplazar un elemento por otro. El aspecto es el módulo utilizado para englobar estas adaptaciones. La figura 6.7 muestra la definición, tomada de [56], de un aspecto de adaptación en `SEAL`. El punto de corte selecciona aquellos elementos que tienen 5 atributos o más, mientras que con el *advice* se reemplazan estas unidades por relaciones, con lo que se

consigue que en lugar de aparecer toda la información del atributo en la página, lo que aparezca sea un enlace a una nueva página, en la que se mostrará esta información. De esta forma, se acorta la información de la unidad principal que se está mostrando en la página.

```
POINTCUT type subunit and contains (count (type attribute) >=5) ;
ADVICE if cm:userDevice = "pda"
          REPLACE hasSubunit BY hasRelationship
```

Figura 6.7: Aspecto de adaptación definido en SEAL

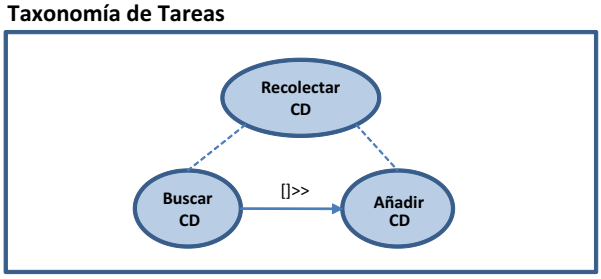
6.2.5 Separación Avanzada de Conceptos en OOWS

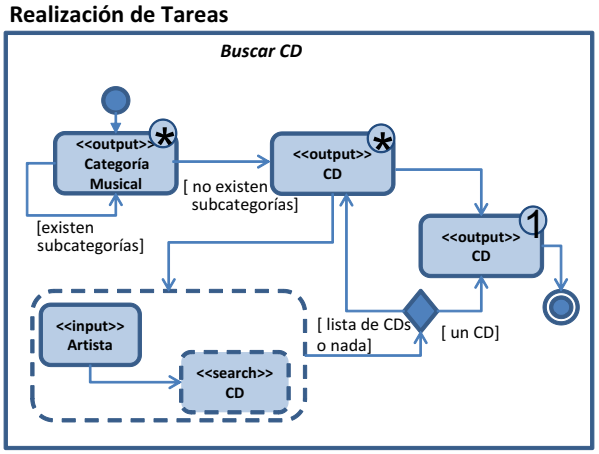
OOWS, inspirándose en la extensión de OOHDM propuesta en [146] también sigue un enfoque simétrico. Como OOWS utiliza una notación basada en la metáfora de tareas para capturar los requisitos; en la extensión [386], los autores usan estas tareas para describir los *concerns*.

Al estar inspirados por [146], se adopta la misma definición de *concern* que la ya explicada en la sección 6.2.3, es decir, se considera que un *concern* es un conjunto de requisitos que tienen que ver con el mismo tema. Los autores se apoyan en la naturaleza interactiva de la Web para proponer como técnica el modelar cada una de las tareas que tienen que ver con los requisitos que están relacionados con el *concern*. Así, cada *concern* se modela dando los siguientes pasos: primero, se define una taxonomía de las tareas que lo componen; segundo, se describe la realización de las tareas; y, finalmente, se especifican los requisitos de información.

En el primer paso, se utiliza la propuesta CTT [301] para modelar las tareas y subtareas que forman parte del *concern*. El segundo paso se basa en el diagrama de tareas obtenido como resultado del primer paso, y describe cómo los usuarios deben realizar aquellas tareas que se han definido como hojas en el árbol que representa la taxonomía de tareas. La descripción se realiza mediante diagramas de actividad UML. Finalmente, se utiliza una plantilla para describir los requisitos de información que dan soporte a las tareas definidas para el *concern*. En la figura 6.8, tomada de [386], se muestran los tres pasos necesarios para describir el *concern* Colección de CDs. En la parte superior izquierda aparece el diagrama de tareas, donde se puede ver cómo la tarea Recolectar CDs está compuesta por dos subtareas: Buscar CD y Añadir CD. En la parte inferior izquierda se muestra el diagrama de realización de tareas para la subtarea Buscar CD. Finalmente, en la parte derecha se muestra una plantilla en la que se indica la información que hay que almacenar para la entidad CD.

A la hora de componer los diferentes *concerns*, los autores vislumbran dos posibles estrategias, o bien hacer la integración a nivel de modelos, o bien hacerla a nivel de





Plantilla con Requisitos de Información

Identificador	O1		
Entidad	CD		
Datos	**Nombre**	**Descripción**	**IP's**
	Título	Título del CD	Output (CD, *) Output (CD, 1)
	Año	Año de grabación del CD	Output (CD, 1)
	Nombre Artista	Artista que ha grabado el CD	Output (CD, *) Output (CD, 1)
	Canciones	Lista de canciones de CD	Output (CD, 1)
	Comentarios	Un breve comentario del CD	Output (CD, 1)
	Carátula frontal	Carátula frontal del CD	Output (CD, *) Output (CD, 1)
	Precio	Precio del CD	Output (CD, *) Output (CD, 1)
	Veces Comprado	Nº de veces que se ha comprado el CD	
	Perfiles Clientes	Perfiles de los clientes que han comprado el CD	

Figura 6.8: Extracto de la descripción del *concern* Colección de CDs.

código. En el caso de esta extensión, los autores abogan por una integración a nivel de modelos, con objeto de poder reutilizar las herramientas que ya tienen definidas en `OOWS`. La integración de modelos de esta propuesta se queda a nivel de requisitos, de tal forma, que se dan mecanismos para integrar los tres tipos de especificaciones que se emplean para especificar *concerns*. Así, hay que realizar una integración a nivel de taxonomía de tareas, una integración de la realización de tareas, y finalmente, una integración de la información de tareas.

La integración de *concerns* a nivel de taxonomía de tareas se realiza añadiendo nuevos arcos que relacionen las tareas del árbol de tareas de un *concern* con las del otro. La integración a nivel de realización de tareas se realiza mediante una serie de reglas de composición, de tal forma que se pueden o bien añadir conexiones entre nodos de dos realizaciones de tareas, o bien mezclar nodos de dos *concerns* distintos. Para hacer la integración a nivel de plantillas, se utiliza un segundo tipo de plantilla, denominado *Plantillas de Integración de Información*, que se asocia con los puntos de interacción de salida (en la figura 6.8 se ven estos puntos de interacción en dos lugares: en el diagrama de realización de tareas son aquellos nodos estereotipados con la palabra `Output`, y en la plantilla de requisitos de información se marca en la columna `IP's`). La idea subyacente en la plantilla de integración es indicar una serie de características con las que se extiende un punto de interacción concreto.

6.2.6 Separación Avanzada de Conceptos en WebDSL

En `WebDSL` la separación de *concerns* se consigue mediante mecanismos lingüísticos (cf. sección 3.3.3), proporcionando distintos sublenguajes para cada uno de los dominios técnicos de la ingeniería web. La integración lingüística de estos sublenguajes [152] asegura la integración de los diferentes aspectos que conforman una aplicación web. Así, se han definido lenguajes para el modelo de datos, definición de interfaz de usuario, el control de acceso [153], los *workflows* [179] y la validación de datos [154]. La figura 6.9, tomada de [152], muestra un ejemplo de los lenguajes para definir el modelo de datos, la interfaz de usuario y la validación de datos. En el recuadro superior, se define la entidad `User` del modelo de datos. En el recuadro central, y mediante el lenguaje de validación se extiende la entidad `User` con las validaciones sobre los atributos. Finalmente, en el recuadro inferior derecho se muestra un ejemplo del sublenguaje de definición de la interfaz de usuario. El recuadro inferior izquierdo muestra una interfaz diseñada con estas características.

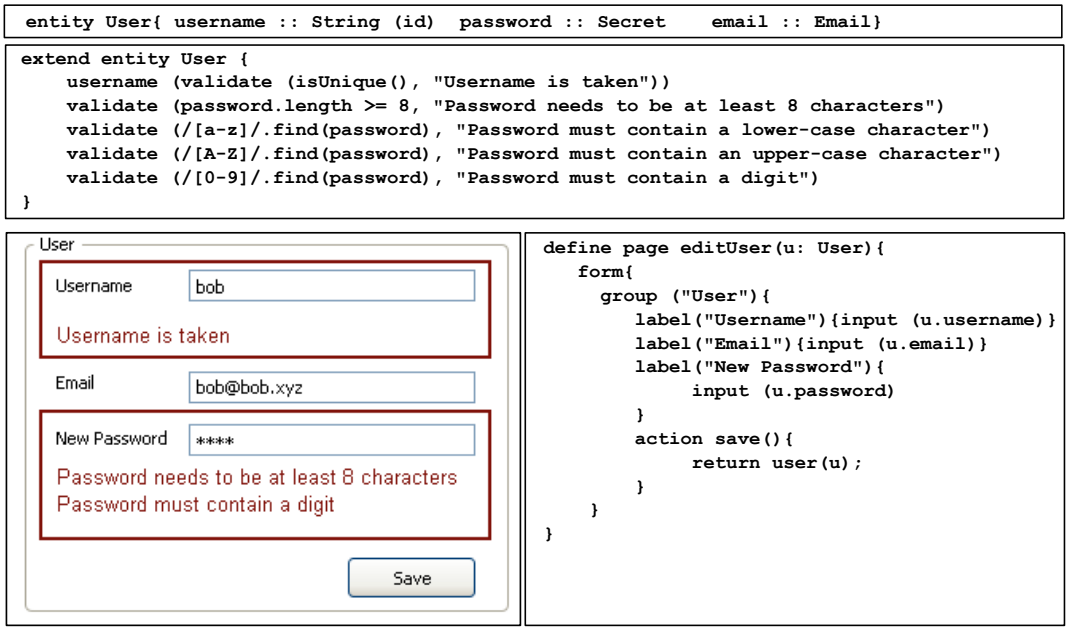


Figura 6.9: Ejemplo de modelo de datos, interfaz de usuario y validación en WebDSL

6.2.7 Orientación a Aspectos en FarNav

Puesto que en `FARNav` la orientación a aspectos fue considerada desde sus comienzos, en este apartado se describirá no una extensión de la propuesta, como ocurría en las secciones anteriores, sino cómo las ideas propuestas por la comunidad de orientación a aspectos se han aplicado en esta propuesta.

La principal aplicación de la orientación a aspectos en `FARNav` se ha realizado a nivel de implementación. En [170] se indica cómo separar y encapsular en aspectos *AspectJ* el código relacionado con el enrutamiento de la navegación en aplicaciones `J2EE`. Por lo tanto, se puede decir que esta propuesta, al hacer uso de *AspectJ*, sigue una aproximación asimétrica.

Básicamente, los autores proponen dos formas de tratar el código del enrutamiento de la navegación en aspectos [171]. Por una parte, está lo que los autores han denominado *Encapsulación*, que consiste en encapsular todo el código relacionado con el enrutamiento de la navegación en un solo módulo de aspectos. Por otra parte, en el segundo enfoque, denominado *Preservar la Estructura de la Aplicación*, no se encapsula todo el código en un módulo, sino que se utilizan varios módulos de aspectos. En la encapsulación, todo el código relacionado con el enrutamiento de la navegación está recogido en un módulo, pero, puede suponer un cuello de botella, ya que todas las peticiones han de pasar por el mismo sitio.

Después de la experiencia con la implementación, los autores concluyen que esta separación a nivel de código no es suficiente, sino que es necesario un modelo que describa explícitamente el enrutamiento de la navegación. En la figura 6.10 se muestra un ejemplo de este tipo de modelo, que los autores denominan Modelo de Enrutamiento de Peticiones [172]. Este modelo permite especificar de forma separada el enrutamiento en las aplicaciones web, por tanto, se puede decir que los autores han definido un lenguaje específico de dominio para modelar de forma separada el enrutamiento de la navegación.

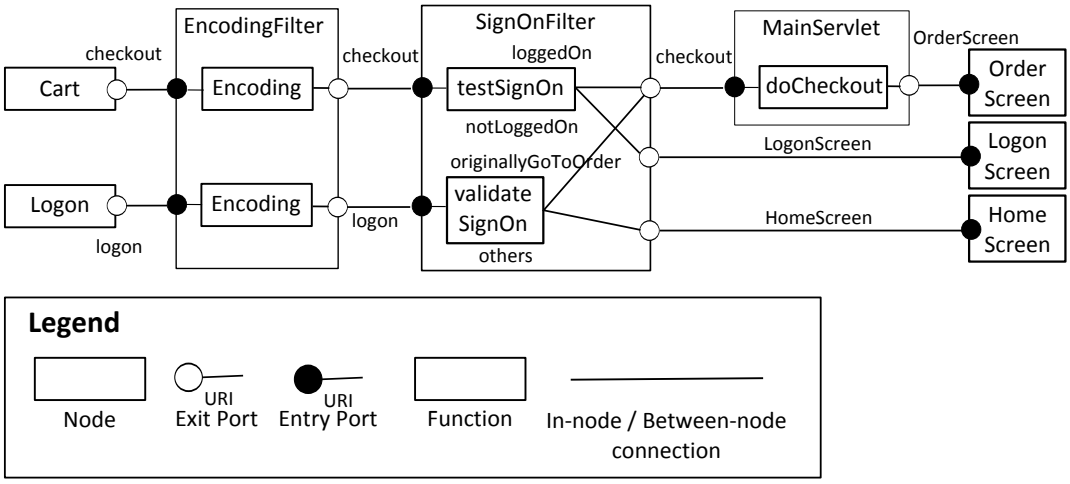


Figura 6.10: Modelo de Enrutamiento de Peticiones en FARNav

6.2.8 Separación Avanzada de Conceptos en WAAT

Al igual que ocurría con `FARNav`, el uso de las ideas de separación avanzada de conceptos en `WAAT` aparece desde su concepción original. En esta propuesta se sigue una aproximación simétrica y su objetivo es realizar minería automática de *concerns* en aplicaciones web. Para conseguir este objetivo los autores se basan en el análisis de conceptos [135], el análisis de impacto [20] y la identificación de *concerns* basada en tokens. Los *concerns* son considerados como aquellas partes del software que son responsables de un objetivo o tarea particular.

Inicialmente, las aplicaciones web se describen mediante modelos orientados a objetos tradicionales, y posteriormente se obtienen un conjunto de trozos de diseño o aplicación (puntos de vista) que sirven para analizar y probar la propia aplicación. Para realizar estas operaciones se siguen tres pasos: en primer lugar, se aplican técnicas de reingeniería para obtener los modelos orientados a objetos que describen la aplicación web; en segundo lugar, se identifican, definen y extraen los *concerns* que componen la aplicación; y, finalmente, se validan los *concerns* obtenidos.

La fase de obtención del modelo orientado a objetos se divide en tres etapas claramente diferenciadas: un análisis del comportamiento de la aplicación, la construcción del modelo, propiamente dicha; y, por último, la validación del mismo modelo. El análisis del comportamiento de la aplicación se realiza mediante el análisis estático y el análisis dinámico de las páginas del sitio web. El análisis estático se realiza para las páginas estáticas y consiste en el análisis del código fuente de la página mediante analizadores léxico/sintácticos. El análisis dinámico consiste en determinar un conjunto significativo de páginas que se generan por el servidor, y generarlas de forma dinámica para analizarlas con posterioridad. Para construir el modelo, los autores utilizan diagramas `UML` tanto de clase como de estado, tal y como se explicó en la sección 5.4.2. La fase de validación es necesaria debido a que en la fase de análisis del comportamiento se utiliza una técnica conocida como *Análisis de mutación*, que puede provocar la introducción en el modelo generado de más información de la que es necesaria. Por lo tanto, la fase de validación consiste en detectar esta información no válida que se haya podido introducir en el modelo.

La identificación de *concerns* se hace desde una perspectiva simétrica, inspirándose en la separación multidimensional de conceptos [293] y en las ideas subyacentes tras *HyperJ* [71]. Para identificar los *concerns* se utiliza la teoría de análisis de datos conocida como Análisis Formal de Conceptos [308], que sirve para identificar estructuras conceptuales dentro de conjuntos de datos. De esta forma, aplicando esta técnica podemos identificar una agrupación de objetos que tienen una serie de atributos comunes. Así, dado un contexto (O, A, R), donde O es un conjunto de objetos, A un conjunto de atributos, y R, una relación binaria entre O y A, los autores definen los *concerns* como las colecciones máximas de objetos que comparten atributos comunes. Para ello, se define una red de *concerns* y se le aplica el algoritmo de Análisis Formal de Conceptos, conocido como `FCA`.

El algoritmo de definición de conceptos consta de los siguientes pasos: primero, se extraen una serie de artefactos del modelo de la aplicación, tales como clases, atributos, métodos, variables, asociaciones, páginas web, enlaces. A continuación, y con ayuda del usuario, se seleccionan una serie de parejas objeto-atributo a partir de este tipo de artefactos (ejemplos de estos pares son las parejas variables-clases o variables-funciones). Luego, se define una matriz I=[clase x clase], conocida como *matriz de impacto*, que contendrá un 1 en el elemento (i,j) si la clase i está relacionada con la clase j y 0 en caso contrario. Después se define una *matriz de contexto* para cada pareja objeto-atributo definida. La matriz de contexto C=[objeto x atributo] es una matriz cuyo elemento (i,j) vale 1 si el objeto i está relacionado con el atributo j, y 0 en caso contrario.

A partir de la matriz de contexto se definen los *concerns* agrupando el número máximo de objetos que tengan atributos comunes. Para visualizar el resultado, los autores utilizan una red de *concerns*, como la mostrada en la figura 6.11. Cada nodo del grafo es un *concern* que agrupa un conjunto de objetos que tienen en común una serie de atributos.

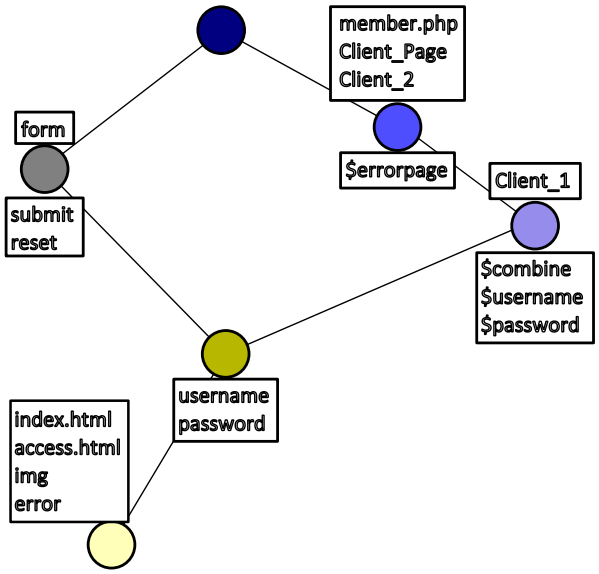


Figura 6.11: Red de *concerns* en WAAT, tomada de [35]

El siguiente paso es la composición de los *concerns* (en el sentido de agrupar aquéllos con propiedades comunes, aunque no desde un mecanismo de composición tal y como se entiende en la orientación a aspectos) y se consigue analizando las dependencias entre los *concerns* para determinar cuáles tienen un comportamiento autónomo y cuáles tienen dependencias de comportamiento. En este paso se utilizan dos técnicas: (1) recorrer el grafo de *concerns* para identificar aquéllos que tienen algún tipo de dependencia y (2) ir agrupando iterativamente *concerns* que comparten atributos.

El último paso del algoritmo consiste en la validación. Los *concerns* obtenidos sirven

para trocear los modelos de las aplicaciones o el código fuente.

6.3 Desarrollo de Software Dirigido por Modelos e Ingeniería Web

El desarrollo de software dirigido por modelos es otra de las tendencias actuales en la ingeniería web. De hecho, todas las propuestas comentadas en el apartado 5.3 han adoptado este enfoque. En este apartado se va a detallar brevemente cómo estas propuestas se han ido ajustando al paradigma DSDM. Además, en el último subapartado se detallan también algunas propuestas que han apostado por este enfoque.

6.3.1 DSDM en WebML

Los conceptos que se manejan en WebML no están definidos mediante un metamodelo explícito, sino que se especifican en términos de Extensible Markup Language (XML) mediante un Document Type Definition (DTD). La herramienta de la que disponen, *WebRatio* [315], también tiene codificado algunos conceptos. Para hacer las transformaciones de modelo a texto se utiliza XSLT, pudiendo generar código Java, JSP, J2EE y *Jakarta Structs*.

Los autores de aspectWebML han definido el metamodelo explícito de WebML para implementar su extensión orientada a aspectos [349]. Para definir este metamodelo, los autores se han basado en el DTD de WebML y han utilizado un enfoque semiautomático de generación de metamodelos MOF a partir de DTD's [350]. Paralelamente, en [258] se presenta también un perfil UML 2.0 para WebML con vistas a poder aplicarle las tecnologías emergentes en el DSDM a esta propuesta.

6.3.2 DSDM en UWE

UWE se basa en UML, tal y como se comentó en la sección 5.3.2. Su principal objetivo es el diseño sistemático y una generación semiautomática de aplicaciones web. Los autores han definido un perfil UML y un metamodelo [226]. Inicialmente, la fase de diseño de UWE estaba soportada por la herramienta *ArgoUWE* [216, 163], una herramienta CASE para el modelado de aplicaciones web, que implementaba el metamodelo de UWE y, utilizando una estrategia de manipulación directa (ver la sección 4.6.2), realizaba una generación semiautomática del modelo de navegación a partir del modelo conceptual, y del modelo de presentación a partir del modelo navegacional.

Actualmente, *ArgoUWE* no se mantiene debido, entre otras cosas, a que al ser diseñado como un *plugin* de *ArgoUML*, solamente se pueden integrar modelos basados en UML 1.4, y no es fácil integrar modelos basados en UML 2.0.

Los trabajos más recientes relacionados con UWE [218, 224, 225] siguen el estándar MDA propuesto por la OMG. Recientemente, han desarrollado un *plugin* para *MagicDraw* [274] y están trabajando con transformaciones modelo a modelo [218]. Además,

las transformaciones en esta propuesta se han definido mediante ATL [94] e implementan la construcción sistemática de una serie de modelos por defecto, que deberán ser refinados por el diseñador. A nivel de CIM, en esta propuesta se utilizan casos de uso con estereotipos tales como <<navigation>> o <<web process>>. A partir de estos casos de uso, los autores han definido una serie de transformaciones modelo a modelo que permiten obtener un primer modelo conceptual por defecto, que el diseñador puede refinar. Del mismo modo, han definido un conjunto de reglas de transformación ATL para obtener un modelo de navegación por defecto a partir del modelo conceptual, y un modelo de presentación por defecto a partir del modelo navegacional.

Para generar una aplicación web a partir de los modelos de diseño, los autores utilizan dos técnicas: por una parte definen una serie de transformaciones ATL para generar el modelo de datos y la capa de presentación de la aplicación; por otra parte, utilizan un enfoque de interpretación para tratar con los modelos de proceso.

6.3.3 DSDM en OOHDM

OOHDMDA [352] es una propuesta basada en OOHDM para generar aplicaciones web basadas en *servlets* que sigue una filosofía MDA. La idea que proponen los autores es construir una aplicación utilizando una herramienta de modelado UML cualquiera, de tal forma que a nivel de PIM se utilicen los modelos OOHDM complementados con la semántica del comportamiento definida en [351]. En *OOHDMDA* se cubren todos los constructores de OOHDM, exceptuando los contextos de navegación, junto con su extensión para tratar con procesos de negocios. Una vez que se ha definido el modelo OOHDM, que contiene clases estereotipadas, y conforma lo que los autores denominan el *PIM Base*, se realizan transformaciones modelo a modelo definidas *ad-hoc*, ya que estas transformaciones se aplican sobre la serialización XMI de los modelos OOHDM. En este proceso de transformación se realizan algunos cambios sobre el modelo original.

El PIM se divide en dos submodelos, el PIM Conceptual y el PIM Navegacional, y se definen otra serie de transformaciones modelo a modelo, realizadas *ad-hoc*, para obtener los correspondientes PSM.

In [271] se definen también una serie de transformaciones para generar el modelo de datos a partir del modelo conceptual. Las transformaciones se definen en OCL.

6.3.4 DSDM en OOWS

OOWS se ha ampliado recientemente con una extensión dirigida por modelos para dar soporte a la integración de procesos de negocio. El modelo de procesos de negocio se especifica mediante una versión extendida de BPMN y sirve de punto de partida para una serie de transformaciones QVT. Las transformaciones se implementan con *Borland Together Architect 2006* para Eclipse. El modelo de negocio en BPMN se transforma en un modelo de navegación por defecto e independiente de la plataforma, con objeto de modelar la interacción con el usuario. Este modelo de navegación tiene que ser refinado por el diseñador, para posteriormente ser transformado a una tecnología web concreta.

6.3.5 DSDM en WebDSL

WebDSL [392] utiliza un enfoque distinto al de las propuestas anteriores. En lugar de usar una visión del Desarrollo de Software Dirigido por Modelos basada en MDA, opta por aplicar la visión de los DSL's. Además, en lugar de trabajar en el espacio tecnológico de los modelos, trabaja en el espacio tecnológico de las gramáticas, de forma que los distintos sublenguajes definidos dentro del marco de WebDSL se especifican con gramáticas. Otra diferencia importante es que los autores de la propuesta han optado por DSL's textuales, en lugar de lenguajes gráficos, como hacían las propuestas anteriores.

La sintaxis de los distintos sublenguajes de especifica mediante SDF [177], mientras que la generación de código se realiza mediante Stratego/XT [44], un lenguaje de transformación de programas basado en reescritura de términos y que permite definir estrategias de reescritura programables. Las reglas se pueden definir sobre la sintaxis concreta o la sintaxis abstracta del lenguaje.

6.3.6 Otras propuestas de Ingeniería Web que siguen una aproximación dirigida por modelos

En este apartado se hace una breve reseña de otras propuestas que también apuestan por un desarrollo web dirigido por modelos.

HyperDE [277] En esta propuesta se define un entorno que combina el DSDM y lenguajes específicos de dominio con el objetivo de hacer un prototipado rápido de una aplicación web. HyperDE permite implementar las aplicaciones web mediante la metodología Semantic Hypermedia Design Method (SHDM) [235], que tiene como objetivo el diseño e implementación de aplicaciones para la web semántica y que comprende cinco actividades: recogida de requisitos, diseño conceptual, diseño de navegación, diseño de interfaz abstracta e implementación. De esta forma, al estar basado en una aproximación relacionada con la web semántica, los autores perciben una aplicación web como una vista navegacional definida sobre una ontología que define el dominio del problema.

HyperDE extiende al *framework* Model View Controller (MVC) y está implementado como una modificación del *framework Ruby on Rails*, en el que la capa de persistencia se ha reemplazado por otra capa basada en *Sesame RDF*. HyperDE genera una serie de DSL's como una extensión de Ruby, que permite la manipulación directa, mediante *scripts* de Ruby, tanto del modelo como del metamodelo de SHDM.

MIDAS [53] En esta propuesta se define un marco metodológico dirigido por modelos para el desarrollo ágil de sistemas de información web basado en MDA. En MIDAS un sistema de información se modela de acuerdo a dos dimensiones. En una dimensión la aplicación se modela desde tres perspectivas distintas: contenido, hipertexto y comportamiento. En la otra dimensión se tienen los diferentes niveles de modelado propuestos por MDA: CIM, PIM y PSM. Los elementos que forman parte del modelo de hipertexto en

MIDAS se definen mediante un perfil UML [57]. Las transformaciones las han descrito de forma textual y las han formalizado con gramáticas de grafos [389].

Moreno et al. [260] El objetivo de esta propuesta es la integración de aplicaciones web con sistemas de terceros, siguiendo la filosofía de MDA. Los autores de esta propuesta definen un *framework* de integración basado en modelos de alto nivel e independientes de plataforma que encapsulan las diferentes abstracciones y mecanismos de interacción. Los autores definen un conjunto de modelos independientes de plataforma divididos en las capas de interfaz de usuario, lógica de negocios y datos. Definen una correspondencia textual entre los conceptos de este modelo y un estereotipo UML. En los trabajos publicados hasta ahora no hay ninguna referencia a las transformaciones, pero los autores planean definirlas usando QVT. Esta propuesta se ha aplicado a CORBA, EJB y RMI [259, 261].

Muller et al. [265] Los autores de esta propuesta siguen una aproximación basada en MDA y definen un metamodelo y una notación asociada para modelar conceptos específicos de web dinámica. Las aplicaciones web se definen a partir de tres modelos distintos: modelo de negocio, modelo de hipertexto y modelo de presentación. En esta propuesta se usa Xion, un lenguaje de acción, para completar y precisar los modelos de negocio y navegación. Tanto para las transformaciones modelo a modelo como para las modelo a texto los autores utilizan el lenguaje MTL [122].

W2000 [24] El objetivo de esta propuesta es el modelado de aplicaciones web complejas. W2000 se puede ver como una evolución de Hypertext Design Model (HDM), pero adaptando esta propuesta a UML y definiendo una notación para procesos de negocio. Un modelo W2000 se organiza en cuatro vistas distintas: *modelo de información*, que define los datos que usará la aplicación y que percibirá el usuario; *modelo de navegación*, indica cómo el usuario navega por la información; *modelo de servicio*, que especifica cómo el usuario puede modificar la información mediante procesos de negocio; y, finalmente, el *modelo de presentación*, que indica cómo los datos y los servicios se le presentan al usuario.

La herramienta de soporte a la propuesta se ha definido como un *plugin* de Eclipse y los modelos se almacenan en *MDR/Netbeans*, un repositorio MOF. El metamodelo de W2000 está definido como un metamodelo MOF [25]. Se utilizan restricciones OCL para asegurarse de que los modelos conformes al metamodelo de W2000 son modelos bien formados. Las restricciones se tratan de dos formas distintas: las topológicas están embebidas en la herramienta, mientras que las de propósito específico se comprueban mediante un validador externo basado en *xlinkit* [272]. Para definir las transformaciones los autores han trabajado con Attributed Graph Grammar (AGG), un lenguaje de transformación basado en grafos.

WebSA [245] El principal objetivo de esta propuesta es cubrir todas las fases de desarrollo de una aplicación web centrándose sobre todo en estrechar la distancia que existe entre los modelos de diseño de las metodologías de diseño web tradicionales y la implementación de la aplicación. En Web Software Architecture (WebSA) una aplicación web está compuesta por ocho vistas agrupadas en tres puntos de vista: requisitos, funcional y arquitectura. El punto de vista arquitectural es la principal contribución a esta propuesta, y está formado por una vista de la arquitectura lógica y una vista de la arquitectura física.

El proceso de desarrollo de WebSA está basado en MDA. A nivel de PIM los autores proponen modelos desde dos perspectivas, la funcional y la de arquitectura. Mediante transformaciones modelo a modelo estas dos perspectivas se integran, para posteriormente aplicarle unas serie de transformaciones modelo a modelo para obtener modelos para plataformas concretas. Para la formalización de las transformaciones los autores usan QVT.

En [244] se proporciona un lenguaje de modelado definido mediante un perfil de UML 2.0 para definir una vista de la arquitectura de las aplicaciones web. Se puede considerar que esta propuesta complementa a otras propuestas de ingeniería web, ya que el punto de vista funcional se puede definir utilizando los lenguajes de modelado de estas otras propuestas.

6.4 Sumario

En este capítulo se ha mostrado cómo se están acoplando las ideas relacionadas con la orientación a aspectos, el desarrollo dirigido por modelos y la ingeniería web. En primer lugar, se ha dado una perspectiva de cómo se están aplicando las ideas de orientación a aspectos en la ingeniería web, para continuar con el desarrollo dirigido por modelos y las propuestas de diseño web. Esta mezcla de áreas se ha tratado de la siguiente manera: en [322, 323] se proponía aplicar ideas de orientación a aspectos al desarrollo web, centrándose en la navegación. En [331, 338] se muestran algunas experiencias con el desarrollo web y la orientación a aspectos. Finalmente, tras analizar las distintas propuestas de lenguajes de modelado para hacer diseños orientados a aspectos, en [332] se concluía que hacía falta elevar el nivel de abstracción, para lo que se proponía utilizar una filosofía basada en MDA. Por último, en [319], se hace un estudio de las distintas propuestas que mezclan el desarrollo de software orientado a aspectos y el desarrollo de software dirigido por modelos, proporcionando un marco homogéneo de comparación y poniendo de relieve algunos problemas de investigación abiertos.

Parte III

Nuestra Propuesta

Do not worry if you have built your castles in the air. They are where they should be. Now put the foundations under them.

Henry David Thoreau

7 Visión General de MWACSL

El objetivo de este capítulo es dar una visión general de `MWACSL`*, de cómo se ha diseñado y de los términos que se manejan en la propuesta. Para ello, el capítulo se distribuye como sigue: en primer lugar, en la sección 7.1 se da una breve introducción, para luego, en la sección 7.2, presentar la arquitectura de una aplicación en* `MWACSL`*. A continuación, en la sección 7.3 se introducen los conceptos que se manejan en la propuesta, tomando como referencia el marco conceptual definido por Stalh y Völter [363] y localizándolos en el mismo. Por último, en la sección 7.4, se concluye el capítulo.*

7.1 Introducción

Uno de los objetivos de esta tesis es proporcionar un marco de trabajo para dar soporte al desarrollo de aplicaciones web teniendo en cuenta varios principios: en primer lugar, se debe aplicar el principio de separación de *concerns*; en segundo lugar, se deben separar también los aspectos dependientes de la tecnología de aquéllos que no dependen de la misma; y, en tercer lugar, los modelos deben ser elementos fundamentales durante todo el proceso de desarrollo.

El primer principio, la separación de *concerns*, se alcanza aplicando las ideas propuestas en el `DSOA` [102, 119], que promulgan nuevas formas de descomponer los sistemas, mejorando la modularización de los mismos para que sean más fáciles de desarrollar, adaptar y mantener [64].

El segundo principio, la separación de aspectos tecnológicos, se alcanza en `MWACSL` gracias a una arquitectura de modelos inspirada en los diferentes niveles de modelado

que se proponen en MDA, la aproximación para el desarrollo del software dirigido por modelos de la OMG. De esta forma, en MWACSL se proporcionan distintas formas de separar *concerns*: por una parte, al aplicar las premisas del DSOA tenemos una separación horizontal (al mismo nivel de abstracción) de los distintos conceptos que conforman un sistema; por otra, al adoptar una perspectiva basada en MDA se consigue una separación vertical, desacoplando los conceptos que dependen de una plataforma concreta de aquellos independientes de la misma (figura 7.2). De esta forma, verticalmente, al igual que en MDA, en MWACSL tenemos distintos niveles de modelado: PIM, PSM[1] y código.

El tercer principio se consigue trabajando en el espacio tecnológico de los modelos, en lugar de en el espacio tecnológico de las gramáticas [228]. Finalmente, y con objeto de elevar el nivel de abstracción de los lenguajes de modelado orientados a aspectos, en MWACSL se aboga a nivel de PIM por lenguajes de modelado de aspectos específicos de dominio, en lugar de lenguajes de modelado de aspectos de propósito general. A la hora de definir estos lenguajes específicos de dominio, se pueden utilizar dos alternativas, bien extender UML mediante perfiles, bien utilizar metamodelos MOF. En [332] se apuntaba como ventaja de la utilización de perfiles el poder usar herramientas genéricas basadas en UML. A día de hoy, y gracias a la proliferación de herramientas de modelado y metamodelado, tales como EMF [49, 311], esto ya no representa un gran inconveniente a la hora de trabajar con metamodelos, teniendo como ventaja su mayor capacidad expresiva.

Teniendo todo esto en cuenta se puede afirmar que los pilares fundamentales en los que se basa esta aproximación son: el desarrollo de software orientado a aspectos, la ingeniería web, el desarrollo de software dirigido por modelos y los lenguajes específicos de dominio para describir *concerns*. De esta forma, el acrónimo MWACSL surge del baile de iniciales de las principales áreas de investigación que se combinan en la propuesta: Model-driven, Web applications, Aspect-oriented and Concern Specific Languages. La figura 7.1 muestra cómo se forma el acrónimo si se colocan los pilares en los que se basa la propuesta en los cuatro puntos cardinales de una rosa de los vientos y se toma como sentido de lectura de las agujas del reloj.

7.2 Arquitectura de modelos en MWACSL

Trabajar con DSL's reporta varios beneficios [388], entre los que se encuentran el proporcionar una representación declarativa, el poder analizar y razonar sobre los conceptos de una forma más simple, el poder comprobar los errores a nivel de dominio, y las optimizaciones. Las principales razones para que la comunidad se decantara por los Lenguajes de Aspectos de Propósito General (LAPGs) en lugar de los Lenguajes de Aspectos Específicos de Dominio (DSAL, usando sus siglas inglesas) son: el importante esfuerzo necesario para implementar un nuevo DSAL; la dificultad de extender el *weaver*

[1] Nótese que las siglas utilizadas corresponden a la nomenclatura inglesa de estos términos, pero se ha decidido utilizar estas siglas porque están mucho más extendidas en la comunidad investigadora.

hecho *ad-hoc* para el DSAL; y, finalmente, la incapacidad de combinar los *weavers* de diferentes DSAL's. Estos tres impedimentos se pueden afrontar si se proporciona una infraestructura apropiada.

Gracias al auge del DSDM, que ha hecho que, por una parte, el interés se centre en los modelos a la hora de desarrollar los sistemas, y, por otra, ha puesto de relieve la importancia de los metamodelos y las transformaciones en el proceso de desarrollo, están surgiendo herramientas cada vez más potentes capaces de dar soporte a la producción de software utilizando este tipo de artefactos. De esta forma, los DSL's han entrado de nuevo en la arena de la comunidad investigadora. Y es que, debido a la infraestructura que proporcionan las técnicas introducidas por el DSDM, los problemas que presentaban los DSALs se pueden solventar más fácilmente, gracias al cambio del espacio tecnológico de las gramáticas (lo que en la bibliografía se conoce como *grammarware*) al espacio tecnológico de los modelos (o *modelware*).

En MWACSL una aplicación estará compuesta por un conjunto de modelos de aspectos o *concerns*, cada uno expresado en un lenguaje específico de dominio. La parte superior de la figura 7.2 muestra un esquema de este tipo de separación. Cada uno de los dominios que se separan a este nivel puede a su vez estar compuesto de distintos *concerns*, si el dominio en cuestión es suficientemente complejo. Un ejemplo de este tipo de dominio es la navegación, que puede estar compuesta de diferentes tipos de *concerns* [197]. En esta tesis nos vamos a centrar en dos *concerns* navegacionales: la navegación estructural y de utilidad y los flujos de navegación.

Al separar los distintos aspectos de un sistema, es más fácil razonar sobre cada uno de ellos, por lo que la complejidad de la aplicación disminuye. Además, al utilizar lenguajes específicos de dominio, los conceptos con los que se va a trabajar van a estar más cercanos al dominio del aspecto o *concern*, de tal forma que cuando el desarrollador trabaja la navegación estructural y de utilidad solamente se tiene que preocupar de términos como nodos, enlaces, navegación principal o navegación local, y cuando trabaja

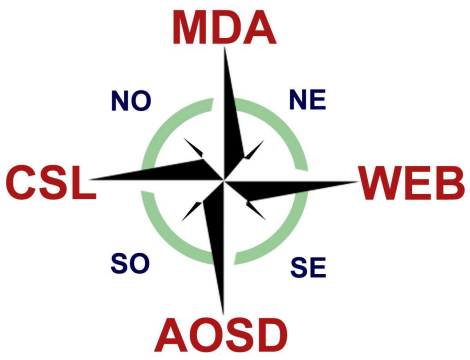


Figura 7.1: Logo de MWACSL

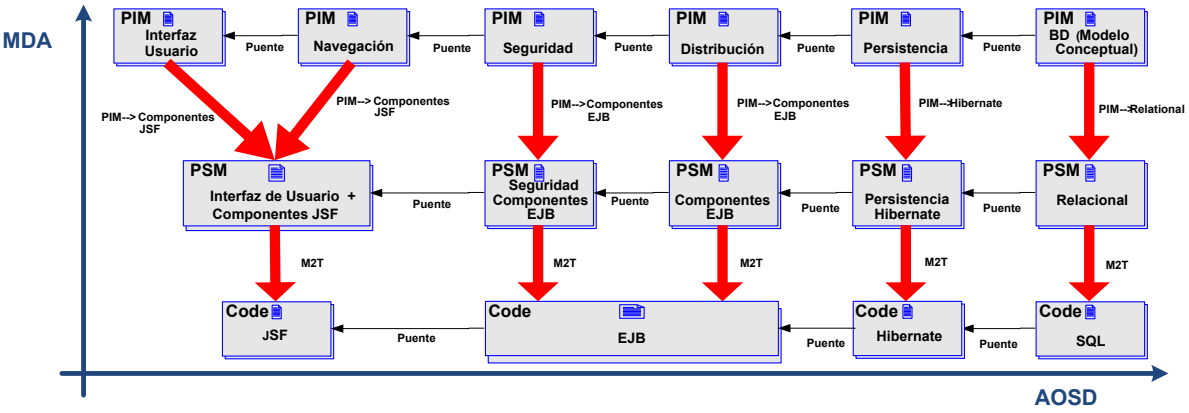


Figura 7.2: Ejemplo de arquitectura de modelos en MWACSL

con el aspecto de sincronización, solamente debe pensar en exclusión mutua, variables de condición, etc.

Lo deseable es mantener estos *concerns* separados desde el diseño hasta la implementación de la aplicación, aunque esto no siempre es posible. Fijémonos, por ejemplo, en la figura 7.2. En ella se necesitan lenguajes específicos de dominio para los *concerns* relacionados con la navegación y con la interfaz de usuario. Sin embargo, como la implementación se va a realizar utilizando el *framework* Java Server Faces (`JSF`) [39], surge un problema, y es que `JSF` no tiene una separación clara entre interfaz de usuario y navegación, por lo tanto, a nivel específico de plataforma, tenemos que trabajar con estos *concerns* mezclados, tal y como se muestra en esta figura. El escoger una plataforma concreta, como en este caso `JSF`, es un tipo de decisión que muchas veces viene impuesto por el propio cliente que ha encargado el producto software.

Para trabajar, por tanto, el diseñador dispondrá de una librería reusable de metamodelos, correspondientes a los diferentes *concerns* que conforman la aplicación. Con el estado actual en el que se encuentra la tecnología, no es difícil integrar estos metamodelos. La combinación de estos metamodelos se verá con más detalles en el capítulo 10.

7.3 Conceptos utilizados en MWACSL

Para explicar los conceptos que se manejan en `MWACSL` nos vamos a basar en el marco conceptual definido por Stahl y Völter [363] para el desarrollo de software dirigido por modelos. Así, este apartado se va a centrar en indicar cómo `MWACSL` encaja dentro del marco conceptual mencionado anteriormente. Los conceptos se explicarán mediante un metamodelo Ecore. Para que sea más fácil el entendimiento este metamodelo y para poder verlo mejor de forma gráfica, los conceptos se van a presentar por grupos, de tal forma que cada uno de los subapartados que se muestran a continuación va a dar más detalles de un grupo de conceptos relacionados. La división en grupos también se ha

realizado basándose en la que usan Stahl y Völter en el capítulo 4 de su libro, aunque se ha añadido un grupo para los conceptos que se usan relacionados con la combinación de *concerns*.

Dentro de cada grupo se distingue entre, por una parte, las clases que forman parte del marco conceptual original propuesto por Stahl y Völter, por otra parte las clases que localizan conceptualmente a `MDA` dentro del *framework*, ya que son conceptos que también se manejan en `MWACSL`, y se usan como forma de separar los conceptos relacionados con las plataformas; y, finalmente, aquellas clases exclusivas de `MWACSL` que están relacionadas, sobre todo, con conceptos que se manejan en el área del desarrollo de software orientado a aspectos. La distinción de estas clases también se ha realizado gráficamente, de forma que cada uno de estos tipos de clases se tiene un color de fondo distinto. Las secciones de metamodelo Ecore se acompañan con una leyenda que especifica la procedencia de cada una de las clases.

7.3.1 Conceptos relacionados con el espacio del problema

La figura 7.3 muestra los conceptos utilizados en `MWACSL` relacionados con el espacio del problema. A continuación se describen los conceptos que forman parte de esta propuesta. Hay que tener en cuenta que no todas las clases que aparecen en la figura tienen una entrada en la lista de descripciones que se detalla a continuación, ya que la semántica de alguna de ellas queda clara en la definición de otra.

Dominio (`Domain`) El término dominio se ha utilizado ampliamente en la ingeniería del software y en el área del desarrollo de software dirigido por modelos, y hace referencia al área de interés o de conocimiento en la que se resuelve un problema. Los dominios pueden definirse o bien con un propósito técnico (como por ejemplo, el definido en [363], cuyo objetivo es describir arquitecturas usando términos como `Entity`, `SystemUseCase`, `Controller` o `Presentation`) o bien para describir los términos de negocio. Los dominios pueden estar compuestos de subdominios más pequeños. Un dominio también puede estar compuesto por una serie de *concerns*.

Concern Un *concern* es un área de interés de un sistema y una forma de descomponerlo en partes más manejables y fáciles de entender. Según la definición de Clarke y Baniassard [60], un *concern* es cualquier funcionalidad en el problema a resolver. Los *concerns* se clasifican en *crosscutting* y *non-crosscutting*. Un *concern* se dice que es *crosscutting* si su representación se mezcla y esparce con la representación de otros *concerns*. En `MWACSL` el dominio de la navegación está compuesto por varios *concerns*, entre los que se encuentran la navegación estructural y de utilidad y los flujos de navegacíón.

Metamodelo (`Metamodel`) El metamodelo es el modo de formalizar la estructura de un dominio o de un *concern*, es decir, de definir los conceptos relevantes de ese dominio y las relaciones existentes entre ellos. Un metamodelo abarca la sintaxis abstracta y la semántica estática de un lenguaje.

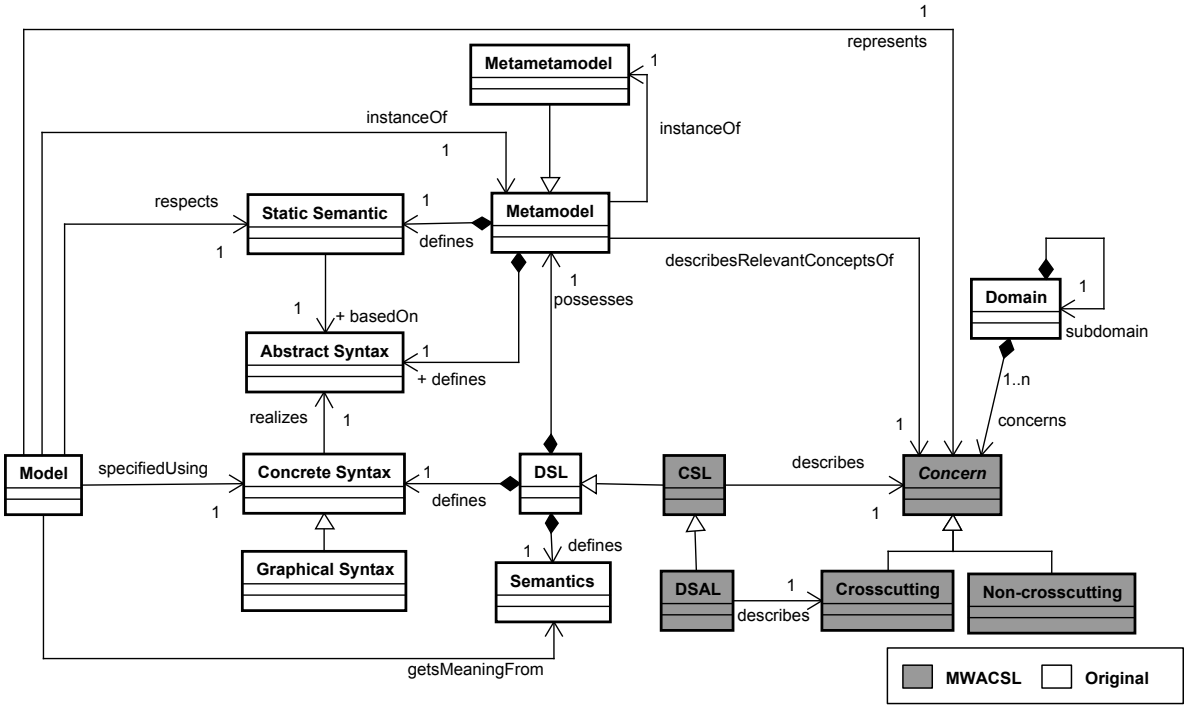


Figura 7.3: Conceptos de MWACSL relacionados con el espacio del problema

Meta metamodelo (`Metametamodel`) Un metamodelo también es un modelo, por lo tanto, debe ser formalizado. Los meta metamodelos definen los conceptos utilizados a la hora de definir los metamodelos. Como los meta metamodelos son también metamodelos, éstos se definen a sí mismos.

Sintaxis abstracta (`Abstract Syntax`) La sintaxis abstracta especifica cuál es la estructura del lenguaje.

Sintaxis concreta (`Concrete Syntax`) La sintaxis concreta es la realización de la sintaxis abstracta. Una sintaxis abstracta puede tener varias sintaxis concretas, por ejemplo, puede tener una sintaxis gráfica o una sintaxis textual.

Semántica estática (`Static Semantics`) La semántica estática de un lenguaje determina los criterios para que los modelos estén bien formados. En los perfiles `UML` o en los metamodelos `MOF` la semántica estática se puede especificar con `OCL`.

DSL Un `DSL` sirve para expresar formalmente todos los elementos claves de un dominio. Para ello, hace uso de un metamodelo, que tiene su sintaxis abstracta y su semántica estática. Además de estos dos elementos, es necesario definir la semántica dinámica para dar significado a los constructores del metamodelo. El término lenguaje de modelado se utiliza muchas veces como sinónimo de `DSL`.

CSL Los dominios pueden estar compuestos de varios *concerns*. Un `CSL` es un `DSL` que expresa formalmente los elementos clave de un *concern*, ya sea *crosscutting* o no.

DSAL Un `DSAL` sirve para expresar formalmente los elementos clave de un *crosscutting concern*. La principal diferencia entre un `CSL` y un `DSAL` es que un `CSL` también abarca *concerns* que no son *crosscutting*. Esto significa que los todos los `DSAL`'s son `CSL`'s pero no al revés.

Semántica (`Semantics`) La semántica (dinámica) es lo que da significado al `DSL` o *concern*, y debe estar bien documentada o ser lo suficientemente clara como para que el modelador pueda definir los modelos correctamente.

Modelo (`Model`) Un modelo es una representación de un sistema, y necesita un `DSL` que indique los elementos claves que se utilizan para la representación. El modelo se formula utilizando la sintaxis concreta del `DSL`, y se puede ver como una instancia del metamodelo. El significado del modelo se obtiene de la semántica del `DSL`.

7.3.2 Conceptos relacionados con el espacio de la solución

En este apartado se describen aquellos conceptos relacionados con el espacio de la solución. Como se puede ver en la figura 7.4, el dominio se realiza a través de una plataforma, que puede seguir la filosofía del `DSOA`, o no.

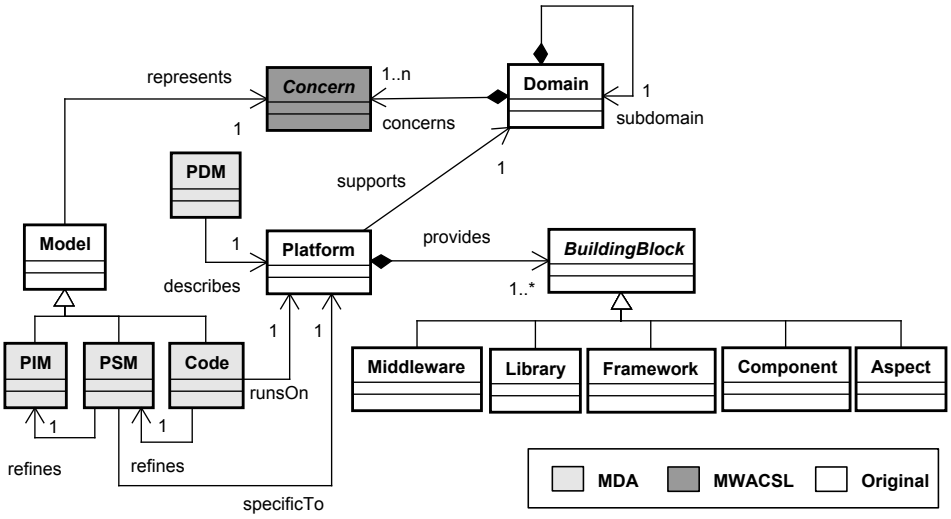


Figura 7.4: Conceptos de MWACSL relacionados con el espacio de la solución

Plataforma (`Platform`) La plataforma se encarga de hacer que la transformación de los modelos sea lo más simple posible, proporcionando la tecnología necesaria para implementar las soluciones. Mientras que el `DSL` describe el espacio del problema,

la plataforma describe el espacio de la solución. Ejemplos de plataformas pueden ser J2EE o Struts. En el contexto de MDA la plataforma se describe mediante un modelo de descripción de plataforma (PDM).

Bloque de construcción (Building Block) Una plataforma se basa en una serie de bloques ya construidos, que pueden ser bibliotecas, *frameworks*, componentes o aspectos.

Modelo independiente de plataforma (PIM) En el contexto de MDA, un PIM es un modelo que describe la lógica de negocio sin ligarse a una plataforma de implementación concreta.

Modelo específico de plataforma (PSM) En el contexto de MDA, un PSM se crea a partir del PIM mediante transformaciones de modelos y es específico para una plataforma, como por ejemplo, J2EE.

Código (Code) En el contexto de MDA, el código es un modelo que se puede ejecutar en la plataforma.

Modelo de descripción de plataforma (PDM) En el contexto de MDA un PDM es un modelo de descripción de plataforma, y es, por lo tanto, un metamodelo de la plataforma objetivo.

7.3.3 Conceptos relacionados con las transformaciones

En este apartado se muestran aquellos conceptos relacionados con las transformaciones. En la figura 7.6 se muestran estos conceptos y las relaciones entre los mismos.

Transformación (Transformation) Una transformación es una clase abstracta. Está basada en un metamodelo origen, y toma como entrada un modelo que es instancia de este metamodelo origen. Se distingue entre transformaciones de modelo a modelo y transformaciones de modelo a texto.

Transformación modelo a modelo (M2M Transformation) Es una transformación cuyo resultado es un modelo. Este modelo tiene que ser conforme a un metamodelo destino. En el contexto de MDA el lenguaje estándar propuesto para este tipo de transformaciones es QVT. Aunque en MWACSL no se restringe el lenguaje de transformación, todas las transformaciones de modelo a modelo de esta tesis se han implementado con QVT.

Transformación modelo a texto (M2T Transformation) Es una transformación que genera código fuente u otros artefactos textuales. En el contexto de MDA el lenguaje propuesto para este tipo de transformaciones es MOF Model to Text.

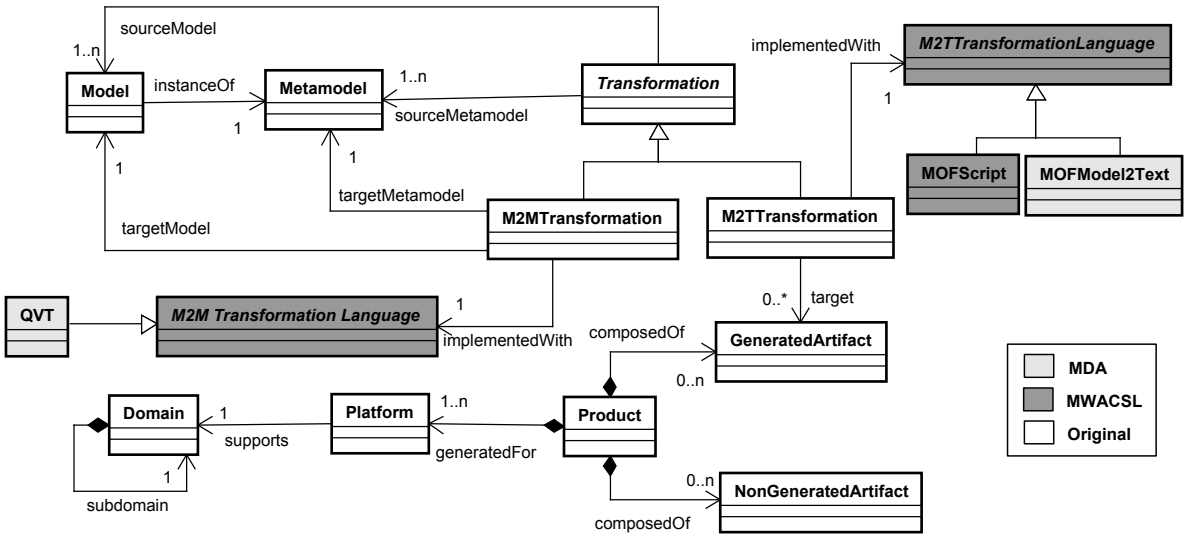


Figura 7.5: Conceptos de MWACSL relacionados con las transformaciones

Producto (`Product`) Al igual que cualquier propuesta basada en el paradigma del `DSDM`, el objetivo de `MWACSL` es crear un producto software, que puede ser una aplicación completa, o un componente para ser usado como bloque de construcción. Con el estado actual de la tecnología es utópico el tener una aplicación generada al 100 %, así que se el producto estará compuesto por una serie de artefactos generados a partir de las transformaciones (`Generated Artifact`) y una serie de artefactos que no son generados (`Non-generated Artifact`).

7.3.4 Conceptos relacionados con la combinación de concerns

Al tener diferentes modelos, cada uno describiendo un dominio o *concern*, se requiere que en algún momento éstos se integren [341]. En `MWACSL` se usa el *weaving* de modelos tal y como se entiende en [50] para especificar las relaciones entre los distintos modelos de *concerns* o dominio, sin embargo, hay que tener en cuenta que en nuestra propuesta no se llega a construir el modelo de todo el dominio al mismo nivel de abstracción en el que están separados los diferentes *concerns*, ya que se ha optado por traducir directamente esta información a la plataforma concreta de implementación.

Modelo de weaving (`Weaving Model`) Un modelo de *weaving* es un modelo que define relaciones (o correspondencias) entre elementos de modelos distintos. La concepción de modelo de *weaving* adoptada en esta tesis es la definida en [84, 85].

Metamodelo de weaving (`Weaving Metamodel`) Un metamodelo de *weaving* es un modelo que define tipos de enlaces [83]. Como no existe un metamodelo capaz de capturar la semántica de todos las posibles relaciones entre los elementos de los

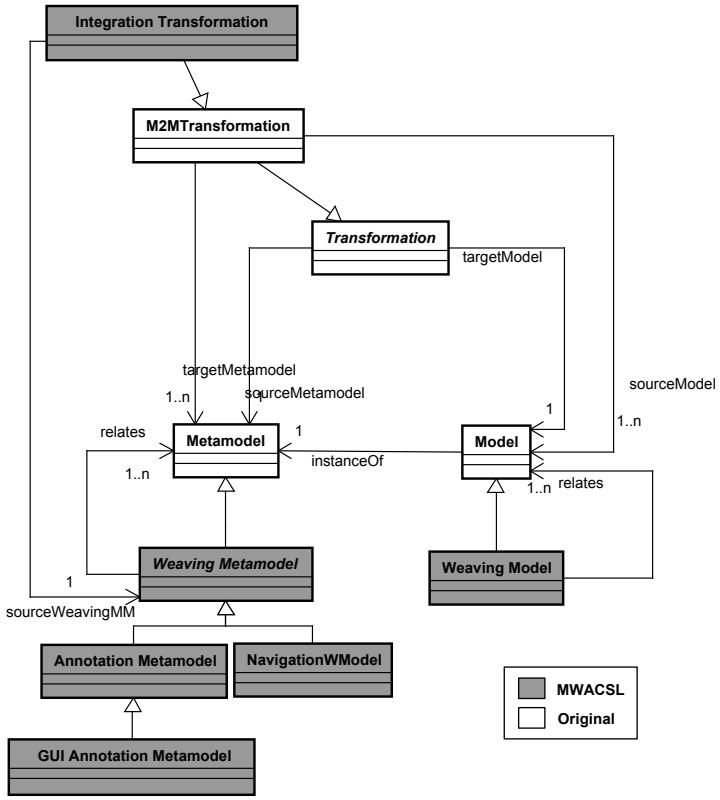


Figura 7.6: Conceptos de MWACSL relacionados con las transformaciones

modelos [51], se usa un metamodelo de *weaving* como un núcleo común, que se extenderá dependiendo de la semántica de los enlaces o relaciones.

Metamodelo de anotación (`Annotation Metamodel`) Un metamodelo de anotación es una extensión al núcleo común del metamodelo de *weaving* que se utiliza para incluir información que no está definida en el metamodelo y que no es conceptualmente relevante para el mismo [83].

Metamodelo de anotación de GUI (`GUI Annotation Metamodel`) Un metamodelo de anotación de GUI es una extensión al metamodelo de anotación definida dentro de `MWACSL` para describir anotaciones que indican con qué tipo de elemento de interfaz se relaciona un enlace de navegación controlada.

Metamodelo de weaving de navegación (`NavigationWMetamodel`) Un metamodelo de *weaving* de navegación es una extensión a un modelo de *weaving* que se usa dentro del contexto de `MWACSL` para definir las relaciones entre dos *concerns* de navegación: la navegación estructural y de utilidad, por una parte, y, por otra, los flujos de navegación.

7.4 Sumario

En este capítulo se ha dado una visión de MWACSL, una propuesta inspirada por varias disciplinas emergentes, tales como el DSOA, el DSDM o la ingeniería web. Si se hace una categorización, y puesto que los modelos son los elementos fundamentales del proceso de desarrollo, se puede considerar que es una propuesta de modelado orientado a aspectos (AOM). En este ámbito, el de las propuestas de AOM, MWACSL se alinea más con aquellas propuestas de modelado de aspectos que se centran en estructurar los modelos de acuerdo a los diferentes aspectos que componen un sistema. Sin embargo, a nivel de PSM no se descartan las propuestas que buscan modelar programas escritos para plataformas orientadas a aspectos. Con respecto a la simetría o asimetría, se puede decir que MWACSL es una propuesta híbrida, ya que permite integración aspecto-base y *concern-concern*.

Además de dar una visión general, y como propuesta también centrada en el ámbito del DSDM, en este capítulo se ha colocado a MWACSL dentro del marco conceptual definido por Stahl y Völter, de forma que se ha explicado la terminología que se maneja en la propuesta.

Por último, cabe destacar que los inicios de esta propuesta se presentaron en [332], mientras que una versión inicial de la integración de aspectos se propuso en [320]. El uso de transformaciones de modelos para tejer aspectos se presentó en [326].

Navigation seems simple, but it's the most subtle and complex part of the interface.

Christina Wodtke and Austin Govella

8

La Navegación desde una Perspectiva Independiente de la Plataforma

Este capítulo se centra en la navegación desde una perspectiva orientada a aspectos, percibiéndola como un dominio que se puede descomponer en diferentes concerns. *Los* concerns *navegacionales que se pueden clasificar en:* concerns *de tareas, basados en temas y "puros" [197]. En este capítulo nos centramos en dos tipos particulares de* concerns *navegacionales, la navegación estructural y de utilidad y los flujos de navegación. De esta forma, el capítulo se estructura como sigue: en la sección 8.2 se especifica un lenguaje específico de dominio para tratar con la navegación estructural y de utilidad. La sintaxis abstracta de este lenguaje se ha especificado mediante el lenguaje de metamodelado* `EMF` *y está detallada en la subsección 8.2.1. Una sintaxis concreta se ha definido en la subsección 8.2.2. Además, en la sección 8.2.3 se definen una serie de transformaciones de modelo a texto que sirven como prueba de concepto de este metamodelo. En la sección 8.3 se define un lenguaje de modelado para tratar con los flujos de navegación siguiendo la misma estructura que en la sección 8.2. Así, en la subsección 8.3.1 se define la sintaxis abstracta, basada en StateWebCharts [411], mientras que en la subsección 8.3.2 se define la sintaxis concreta. Finalmente, se resume el capítulo.*

8.1 Introducción

Los cambios de tecnología y de requisitos son más frecuentes en las aplicaciones web que en las aplicaciones software tradicionales. Uno de los aspectos más destacables y distintivos de las aplicaciones web es la navegación que, aunque parezca simple, es una de las partes más sutiles y complejas de la interfaz [413]. Algunas de las preguntas no técnicas a las que hay que responder cuando se diseña la navegación de un sitio web son: ¿qué información se ha de incluir en el sitio web?, ¿cómo se organizará esta información? o ¿cómo pueden los usuarios encontrar la información que buscan? A estas preguntas se les intenta dar respuesta desde dos disciplinas distintas, la Arquitectura de la Información (`AI`) y la Ingeniería Web (`IW`). Mientras que la `AI` está más relacionada con organizar, estructurar y etiquetar la información, el diseño de la navegación dentro de la `IW` tiene más que ver con la parte técnica y visual. Sin embargo, existe una clara desconexión entre ambas disciplinas [22], lo que provoca que se obtengan malos diseños bien porque la tecnología de la información está muy separada del proceso de diseño, bien porque se ha integrado demasiado tarde en el mismo.

La navegación, por tanto, es un aspecto sutil, pero complejo, e importante dentro del diseño de un sitio web, al que se le han dedicado tanto varios capítulos de libros [414, 403, 275, 366] como libros completos [206]. Al ser la navegación un dominio complejo, se puede descomponer en un conjunto de *concerns*. Un *concern* navegacional [197] es aquél que se manifiesta en la estructura navegacional de la aplicación y que impacta en el modo en el que los usuarios navegan por la misma. Además, los *concerns* navegacionales se reflejan en alguna estructura de información, de navegación o en algún comportamiento de la navegación, impactando, por tanto, en la estructura navegacional de la aplicación.

Los *concerns* navegacionales se pueden clasificar en: *concerns* de tareas, *concerns* basados en temas y *concerns* "puros" [197]. Los *concerns* de tareas están relacionados con las diferentes acciones de alto nivel que puede realizar un usuario en una aplicación web. En algunos sitios a estas tareas de alto nivel se les llama flujos web. Un ejemplo de este tipo de *concern* puede ser el carrito de la compra que se usa en muchas aplicaciones de comercio electrónico. Los *concerns* basados en temas suelen aparecer en sitios puramente informacionales. Un ejemplo de este tipo de *concern* puede ser el género de un libro mientras se realiza una búsqueda en `Amazon.com`. Finalmente, los *concerns* navegacionales "puros" son abstracciones usuales en el diseño de la navegación como, por ejemplo, las visitas guiadas.

En este capítulo nos centramos en dos tipos particulares de *concerns* navegacionales: la navegación estructural y de utilidad y los flujos de navegación web. El primero de ellos puede ser clasificado como un *concern* navegacional "puro", mientras que el segundo es de los de tareas. Para cada uno de estos *concerns* se ha definido un lenguaje específico de dominio. El diseño de estos lenguajes se ha realizado fijándose tanto en la `AI` como en la `IW`. Así, el lenguaje definido para modelar la navegación estructural y de utilidad se inspira en los mapas de sitio web, un entregable usado ampliamente en el ámbito de la `AI`, mientras que el lenguaje definido para tratar los flujos web se ha inspirado

en StateWebCharts [411]. Ambos lenguajes se han definido sin ningún detalle de la plataforma en la que se van a implementar.

8.2 Navegación estructural y de utilidad

La estructura de un sitio y la navegación están relacionadas, pero no son lo mismo. Un mapa de un sitio web detallado muestra todas las páginas del sitio, pero la navegación es una vista limitada sobre esta estructura. La navegación estructural define la estructura o jerarquía del sitio web. Para modelar esta navegación, nos basamos en los mapas del sitio web (*sitemap*), uno de los entregables más usados en el ámbito de la Arquitectura de la Información. La arquitectura de la información [361] es una de las disciplinas que se encarga de, entre otras cuestiones, ayudar a los usuarios de aplicaciones a encontrar el camino hacia la información que están buscando, es decir del diseño de la navegación.

Existen dos tipos de navegación estructural, la navegación principal y la navegación local. Mientras que con el primer tipo de navegación estructural se representan a las páginas situadas a más alto nivel, con el segundo se accede a los niveles más bajos en la jerarquía del sitio. Normalmente, la navegación principal aparece en todas las páginas del sitio; la navegación local, sin embargo, trabaja en conjunción con la principal y se trata de forma diferente a ésta porque suele variar con más frecuencia. En el metamodelo definido en este apartado, se distingue entre estos dos tipos de navegación estructural, ya que se comportan de forma algo diferente.

Como la navegación de utilidad se comporta de forma similar a la navegación principal, en el sentido de que se acaba traduciendo en un grupo de enlaces que aparecen en todas las páginas del sitio web, se ha decidido incluir en el metamodelo la navegación de utilidad, situándola fuera de la jerarquía del sitio. De esta manera se amplía el conjunto de elementos utilizados en los mapas de sitio más básicos. Finalmente, otra diferencia con los mapas del sitio básicos es el uso de referencias como mecanismo para facilitar la legibilidad y mejorar su escalabilidad. Cuando los sitios web son muy grandes y tienen muchas secciones (véase, por ejemplo, `Amazon.com`), es común que a una sección concreta se acceda desde caminos distintos, para facilitar el acceso a la misma. A efectos del mapa de sitio, esto provoca que se entrecrucen muchas líneas en el mismo, pasando de verse gráficamente como un árbol a verse como un grafo. Para evitar que el propio mapa del sitio quede enmarañado, lo que usamos es el concepto de referencia, de forma que el aspecto visual del mapa sigue quedando como un árbol y un nodo aparecerá repetido varias veces en el mismo. La referencia se distinguirá del propio nodo gráficamente.

Como entregables, los mapas del sitio definen la navegación estructural en los sitios web y una de sus principales aportaciones es que permiten a los diseñadores anticiparse a los errores de organización de la información. Pero, a pesar de su popularidad, no hay una práctica estándar a la hora de definir los mapas del sitio web. El mapa del sitio más básico [48] es similar a un diagrama de organización en el que las cajas o rectángulos representan páginas o áreas del sitio, y también pueden representar navegación, aunque

no tienen por qué. Los rectángulos están conectados por líneas, que representan relaciones entre contenido sin sugerir una navegación explícita. El sitio en el que se colocan las páginas y las conexiones, indican que existen una jerarquía entre páginas. La figura 8.1 muestra un ejemplo de este tipo de entregables, al que se le ha añadido el tratamiento de la navegación de utilidad; las referencias se distinguen del los nodos apareciendo con fondo gris.

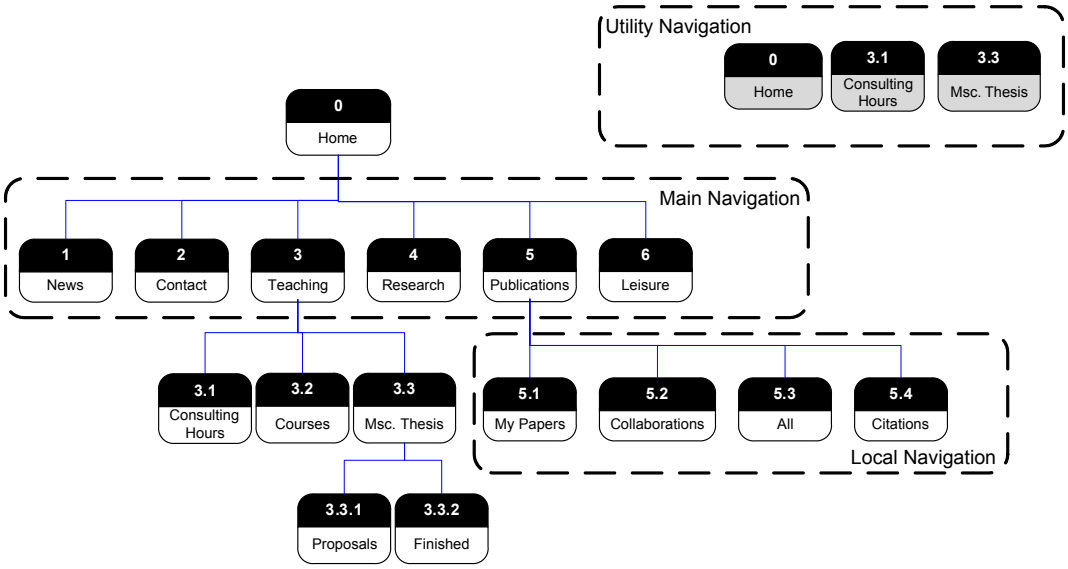


Figura 8.1: Un ejemplo de mapa del sitio web

8.2.1 Metamodelo

Además de considerar el tratamiento conjunto de la navegación estructural y de utilidad, a la hora de diseñar el lenguaje de modelado, se han realizado dos suposiciones que restringen la semántica de mapa del sitio definida anteriormente:

1. No se van a modelar todas las páginas del sitio, sino solamente aquéllas relacionadas con la navegación estructural y de utilidad.

2. Solamente se van a representar un conjunto mínimo de los elementos comunes a las diferentes representaciones de los mapas del sitio definidas en [206], de forma que el lenguaje de modelado sea lo más sencillo posible.

Para garantizar estas suposiciones, se ha propuesto el metamodelo representado en la figura 8.2, mediante el que se define la sintaxis abstracta del lenguaje de modelado. Este metamodelo es una evolución de los publicados en [328, 329]. El elemento raíz es la metaclase `Sitemap`. Un mapa del sitio está compuesto por uno o más elementos y

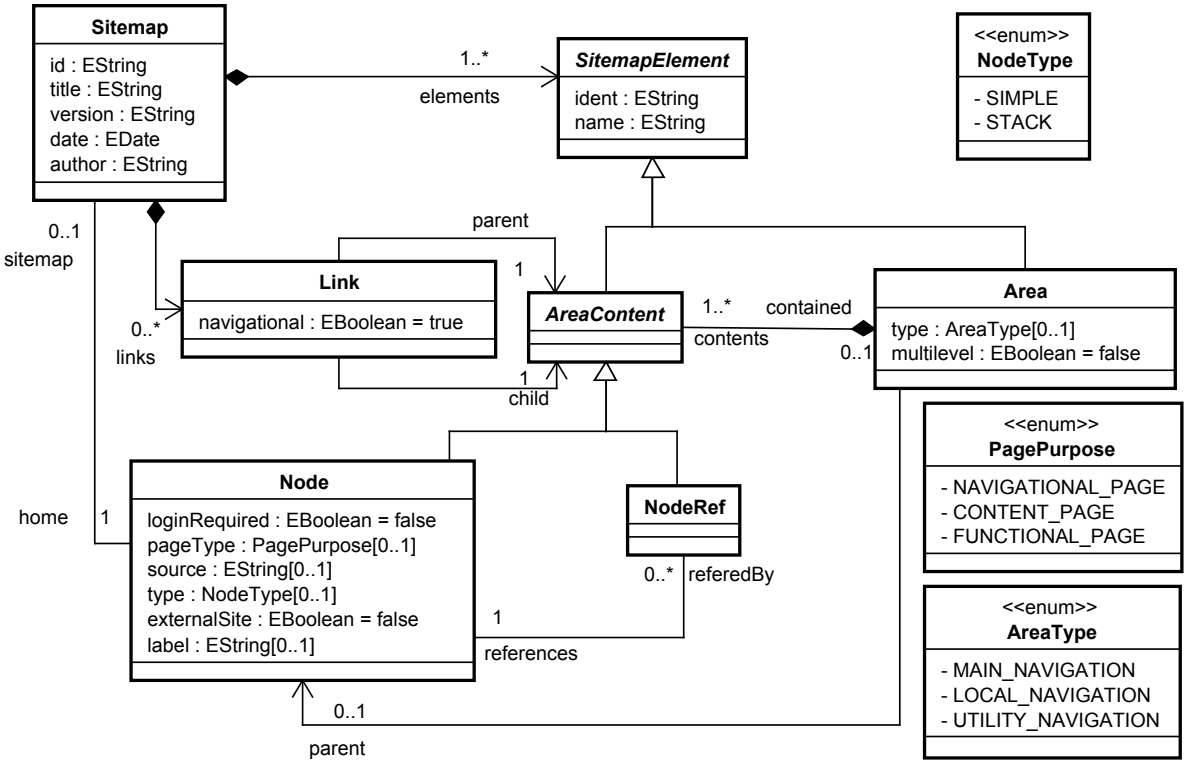


Figura 8.2: Metamodelo que representa un mapa de sitio web

cero o más enlaces. Esto se representa mediante las referencias `elements`, que relaciona `Sitemap` con la clase abstracta `SitemapElement`, y `links`, que enlaza `Sitemap` con `Link`. La clase abstracta `SitemapElement` modela las propiedades que son comunes a todos los elementos de un mapa del sitio: un identificador único, recogido en el atributo `ident`, y un nombre (`name`).

Principalmente hay dos tipos de elementos en un mapa de sitio: `Area` y `AreaContent`. `Area` modela un conjunto de nodos que tienen una función de navegación similar, mientras que `AreaContent` es una clase abstracta que representa los elementos que pueden estar contenidos en un área: nodos y referencias a nodos. Hay que señalar, sin embargo, que estos dos tipos de elementos no siempre han de estar contenidos en un área, ya que pueden aparecer en el mapa del sitio sin estar incluidos en un área concreta. Éste sería el caso, por ejemplo, del nodo `3.2` de la figura 8.1. `Area` tiene dos propiedades: `type`, que representa la función de navegación del área; y `multilevel`, que indica si el área contiene nodos situados a varios niveles de profundidad. Un ejemplo de este tipo de área es la navegación principal de `Amazon`, que para estructurar la información muestra un menú con varios niveles, aunque solamente los niveles inferiores conducen a secciones del sitio (para más información vaya a la sección 11.5.1).

Un nodo (`Node`) representa una página o conjunto de páginas relacionadas del sitio web (como un *hub* de páginas de productos paginadas), mientras que `NodeRef` es una referencia a un nodo. El identificador de un nodo normalmente es un esquema de numeración jerárquico, que indica la profundidad del nodo dentro del sitio. Además, un nodo tiene las siguientes propiedades: `loginRequired`, que indica si el usuario tiene que estar autentificado para acceder al nodo; `pageType`, que indica el tipo de página que representa; `source`, que recoge la `URL` en la que se sitúa el nodo; `type`, que modela si el nodo representa una página simple o un conjunto de páginas; `externalSite`, que indica si el nodo representa una referencia a un sitio externo; y, `label`, que modela una etiqueta con el nombre que aparecerá en el sitio web asociado al enlace que conduce al nodo.

Las áreas y los nodos (o sus referencias) pueden tener padre o no. La relación paterno-filial entre nodos y referencias a nodos se refleja en la metaclase `Link` y las referencias `parent` y `child`, mientras que el padre de un área se representa mediante la referencia `parent` que relaciona `Area` y `Node`. Las áreas que no tienen padre son aquéllas que representan la navegación de utilidad. Este hecho se ha modelando mediante una restricción `OCL`, teniendo como objetivo que el metamodelo quede lo más simple posible (ver listado 8.2). La propiedad `navigational` de `Link` modela una relación paterno-filial que no implica navegación, como en el caso de las área multinivel. Con relación a los nodos, el nodo raíz del mapa del sitio se ha modelado mediante la relación `home` entre `Sitemap` y `Node`.

Un nodo puede acceder a los `NodeRef` que lo referencian mediante la relación `referedBy` (la opuesta a `references`). Además, un área puede contener uno o varios nodos o referencias a nodos (usando la referencia `contents`). En la figura 8.1 se han representado los tres tipos de áreas que consideramos en nuestro metamodelo y que se han etiquetado como `MAIN_NAVIGATION`, `LOCAL_NAVIGATION` y `UTILITY_NAVIGATION`.

Para complementar a las metaclases, también se han definido tres enumeraciones, dos de ellas relacionadas con los nodos (`PagePurpose` y `NodeType`) y la otra relacionada con las áreas (`AreaType`).

`PagePurpose` especifica el objetivo principal de un conjunto de nodos. La clasificación en tres grandes categorías se ha tomado de [206]. De esta forma, existen páginas navegacionales, páginas de contenido y páginas funcionales. Las páginas navegacionales (`NAVIGATIONAL_PAGE`) tienen como objetivo el indicarle al usuario el camino hacia su destino final. Las páginas de contenido (`CONTENT_PAGE`), como su propio nombre indica, son las páginas que contienen la información que el usuario va buscando. Finalmente, las páginas funcionales `FUNCTIONAL_PAGE` son aquéllas que le permiten al usuario completar una tarea.

`NodeType` indica si el nodo representa una página simple (`SIMPLE`) o un conjunto de páginas similar (`STACK`). Los nodos que pertenecen a un área tienen un objetivo de navegación común. Con `AreaType` se modelan: la navegación principal (`MAIN_NAVIGATION`), la navegación local (`LOCAL_NAVIGATION`) y la navegación de utilidad (`UTILITY_NAVIGATION`).

La navegación principal es el menú o conjunto de enlaces que aparecen en cada página del sitio web y que enlaza con las principales secciones del sitio. El comportamiento de la navegación de utilidad es parecido al de la navegación principal, ya que se utiliza para complementarla, conteniendo enlaces que al usuario le gustaría tener en cada página, pero que normalmente no tienen el mismo peso visual que la navegación principal. Finalmente, la navegación local guía al usuario a ciertas secciones dentro de una sección mayor.

8.2.1.1 Restricciones OCL

El metamodelo definido en la figura 8.2 se acompaña con una serie de restricciones especificadas en `OCL`. Estas restricciones se han implementado usando el *plugin* de Eclipse para `OCL` incluido dentro del proyecto de modelado Eclipse `EMP`. Algunas de estas restricciones se han introducido para mantener simplificado al máximo el metamodelo. A continuación, se especifican agrupadas por el contexto al que se aplican.

Contexto Sitemap. Dentro del contexto de la metaclase `Sitemap` se han definido las siguientes restricciones (listado 8.1):

Listado 8.1: Restricciones OCL asociadas al contexto Sitemap

```
1 context Sitemap
2 inv: elements->isUnique(ident)
3 inv: not links->exists(l | l.child = home)
4 inv: home.parent.oclIsUndefined() and home.parent = NodeType::SIMPLE
    and home.parent = PagePurpose::FUNCTIONAL_PAGE
5 inv: elements->select(oclIsTypeOf(Area))->select(a | a.oclAsType(Area).
    type = AreaType::MAIN_NAVIGATION)->size() = 1
```

1. (Línea 2) Un nodo solamente puede aparecer una vez en el mapa del sitio. Si ha de estar en varios sitios, entonces se han de usar referencias al mismo. Esto implica que no pueden existir dos nodos con el mismo identificador. Esta restricción la ampliamos evitando que existan dos elementos en el *sitemap* con el mismo identificador.

2. (Línea 3) No existe ningún `Link` cuyo hijo sea el nodo `home`, por lo tanto, el nodo raíz del sitio no puede ser hijo de ningún otro nodo.

3. (Línea 4) El nodo raíz del mapa del sitio no tiene padre; su propiedad `pageType` debe ser `FUNCTIONAL_PAGE` y su tipo `SIMPLE`.

4. (Línea 5) Solamente puede haber un área de navegación principal.

Contexto Area. Dentro del contexto de la metaclase `Area` se han definido las siguientes restricciones (listado 8.2):

Listado 8.2: Restricciones OCL asociadas al contexto Area

```
1 context Area
2 inv: type = AreaType::UTILITY_NAVIGATION implies
3             parent.oclIsUndefined()
4 inv: type = AreaType::MAIN_NAVIGATION implies
5             not parent.oclIsUndefined()
6 inv: type = AreaType::LOCAL_NAVIGATION implies
7             not parent.oclIsUndefined()
8 inv: let refs: Collection(NodeRef) = (contents->select(ac|ac.
    oclIsKindOf(NodeRef)))->collect(oclAsType(NodeRef)) in (contents->
    union(refs->collect(references)))->isUnique(ident)
```

1. (Línea 2) Las áreas que representan navegación de utilidad, no tienen padre, puesto que están fuera de la jerarquía del mapa del sitio.
2. (Líneas 3 y 4) Las áreas de navegación principal y navegación local han de tener un padre.
3. (Línea 5) En un área no puede haber dos elementos que hagan referencia al mismo nodo. Esta restricción es un poco más compleja, y necesita alguna explicación más. En primer lugar se obtiene la colección de todas las referencias a nodos que estén incluidas en el área a comprobar (que se almacena en la variable `refs`). Luego se realiza la unión de todos los elementos contenidos en el área con la colección de nodos a los que referencian las referencias incluidas en `refs`. Si en este conjunto hay nodos duplicados, significa que hay más de una referencia al mismo nodo.

Contexto Node. Dentro del contexto de la metaclase `Node` se han definido las siguientes restricciones (listado 8.3):

Listado 8.3: Restricciones OCL asociadas al contexto Node

```
1 context Node
2 inv: contained.type = AreaType::MAIN_NAVIGATION implies referedBy->
    exists(nr |not( nr.contained.oclIsUndefined()))
3 inv: externalSite implies source.size() > 0
```

1. (Línea 2) Los nodos que pertenezcan al área de navegación principal no pueden estar en otras áreas. Los enlaces que pertenecen al área de navegación principal tienen que aparecer en la mayoría de las páginas del sitio web, por lo que no tiene sentido que los nodos del área de navegación principal aparezcan en otras áreas, ya que aparecerían duplicados. Como anteriormente se ha puesto la restricción de que un nodo solamente puede aparecer una vez (línea 5 del listado 8.2), la comprobación solamente hay que hacerla para las referencias a los nodos.

2. (Línea 3) Si un nodo hace referencia a un sitio externo, entonces hay que indicar su `URL` en el atributo `source`.

Contexto Link. Dentro del contexto de la metaclase `Link` se han definido las siguientes restricciones (listado 8.4):

Listado 8.4: Restricciones OCL asociadas al contexto Link

```
1 context Link
2 inv: parent <> child
```

1. (Línea 2) Un nodo no puede ser padre de sí mismo.

8.2.2 Una sintaxis concreta

En la sección anterior se ha definido lo que se puede considerar como la sintaxis abstracta del lenguaje. Para la sintaxis concreta se ha definido un lenguaje gráfico inspirado en la representación que se utiliza en el área de la Arquitectura de la Información para describir el mapa de un sitio web.

Como no existe una práctica estándar para representar los mapas de sitio web, se han minimizado los elementos del lenguaje gráfico, de forma que se representan los nodos, las referencias a nodos, las áreas y las relaciones padre/hijo entre nodos. La figura 8.3 muestra los elementos gráficos empleados para representar los nodos, las referencias a nodos, las áreas y los enlaces, respectivamente. Para simplificar los modelos, solamente aparecen visibles algunas de las propiedades de los nodos y de las áreas. En concreto, para un nodo se muestran su identificador (`id`) y su etiqueta (`label`), para una referencia a nodo, también se muestran el identificador y la etiqueta, pero del nodo al que está referenciando y, finalmente, para un área se representan su identificador (`id`) y su tipo (`type`). Una relación entre dos nodos se representa con una línea continua que los une. La línea tiene uno de los extremos acabados en una punta de flecha. La la flecha parte del nodo padre y se dirige hacia el nodo hijo.

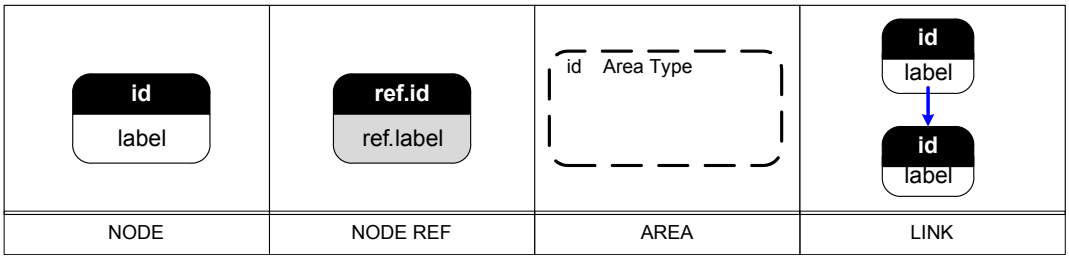
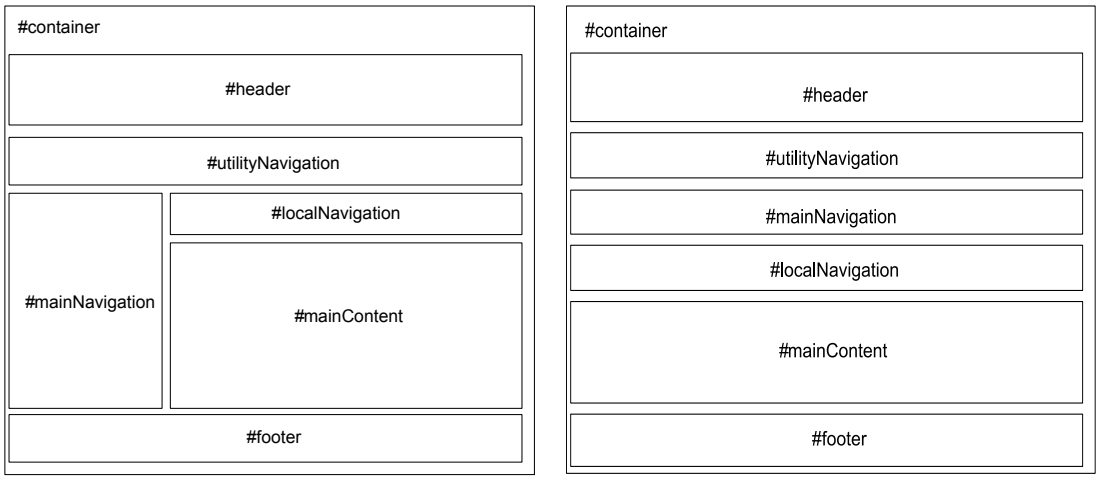


Figura 8.3: Elementos gráficos asociados a los elementos del mapa del sitio

8.2.3 Una prueba de concepto mediante transformaciones de modelo a texto

Como forma de tener una referencia visual de la navegación definida por los modelos conformes al metamodelo definido en la sección 8.2.1, se han definido una serie de transformaciones de modelo a texto que generan un esqueleto `HTML` que muestra exclusivamente la navegación estructural y de utilidad para un sitio web, con el objetivo de poder percibir la estructura del sitio sin distraerse con el contenido del mismo. Como se indica en el título de este apartado esto es una prueba de concepto.

Las transformaciones se basan en la idea de que, objetivamente, la navegación estructural y de utilidad se pueden ver como un conjunto de enlaces a varias páginas de un sitio web [149]. De esta forma, cada tipo de navegación generará una lista de enlaces `HTML`, a la cual se le aplicará un estilo mediante un archivo de hojas de estilo `CSS`. El estilo a aplicar dependerá del tipo concreto de navegación. Por ejemplo, la figura 8.4 muestra dos distribuciones comunes de la navegación principal y local. En la parte izquierda de la figura 8.4(a) se muestra una distribución que se conoce como distribución en L invertida y coloca los enlaces del área de la navegación principal en una distribución vertical y la navegación local horizontalmente, de manera que ambas forman la figura de una L invertida. En la parte derecha de la figura 8.4(b) la distribución de ambos tipos de navegación es horizontal, apareciendo la navegación principal por encima de la local. Tanto para una distribución como para otra, el conjunto de enlaces generados será el mismo, lo que dará la apariencia de una distribución u otra será el archivo de hoja de estilo que se le aplique.



(a) Distribución en L invertida. (b) Distribución horizontal

Figura 8.4: Distribuciones asociadas a las hojas de estilo

Así, por cada nodo del mapa del sitio se generará un archivo `HTML`. Dicho archivo contendrá una serie de listas con enlaces, dependiendo del tipo de navegación que se haya definido. Para las distribuciones de la figura 8.4 se tendrían que generar tres listas de

enlaces, para `utilityNavigation`, para `mainNavigation` y para `localNavigation`.En la figura 8.5 se muestra uno de los archivos `HTML` generado con las transformaciones y tomando como modelo de entrada el mapa de la figura 8.1. Concretamente, este archivo es el que se genera para el nodo cuyas propiedades `id` y `name` toman los valores `5.3` y `All`, respectivamente. El nombre del archivo generado está compuesto por el identificador del nodo, seguido de un guión, el nombre y la extensión `.html`. Fíjese que existe una pequeña transformación en el nombre del archivo, ya que se sigue el convenio para nombrar a los archivos propuesto en [149], mediante el que se aconseja nombrar a los archivos `HTML` en minúsculas y sin usar espacios en blanco. Así, para un nodo cuyo atributo `name` tenga el valor `All` (como sería el caso del nodo `5.3` de la figura 8.1) se generara un archivo `HTML` cuyo nombre será `5.3-all.html`. Si hubiera algún espacio en blanco, como ocurre en el nodo con identificador `5.1` de la figura 8.1, éste se sustituiría por un guión.

Figura 8.5: Archivo 5.3-all.html generado para el nodo con id 5.3

Las transformaciones descritas en este apartado se han implementado en MOFScript [312]. Si quiere más detalle de las mismas, o ver el código fuente, vaya al sitio web de `MWACSL`[1] . En el resto de la sección se dan más detalles de las mismas, centrándose en cada uno de los tipos de navegación descritos anteriormente.

8.2.3.1 Transformando la navegación principal

La navegación principal [367] es una navegación que aparece en todas las páginas de un sitio web y que enlaza a las principales áreas del mismo. Esto implica que la lista `HTML` que se genera para este tipo de navegación debe incluirse en cada uno de los archivos `HTML` que se generan. En la figura 8.6 se muestra el trozo de código `HTML` que se genera para la navegación principal definida en el mapa de la figura 11.5.

Además de mostrar el trozo de código generado a partir de la transformación, la figura también muestra cómo puede cambiar el aspecto visual de la navegación principal al aplicar distintos estilos. En concreto, mediante el estilo definido en el archivo `InvertedLMainNav.css` se presenta la navegación como un menú vertical, mediante el definido en `HorizontalMainNav.css` se presenta como una barra horizontal, y con el definido en `TabbedMainNav.css` se muestra como una serie de pestañas.

[1] `http://www.lsi.us.es/~reinaqu/org.mwacsl/html/`

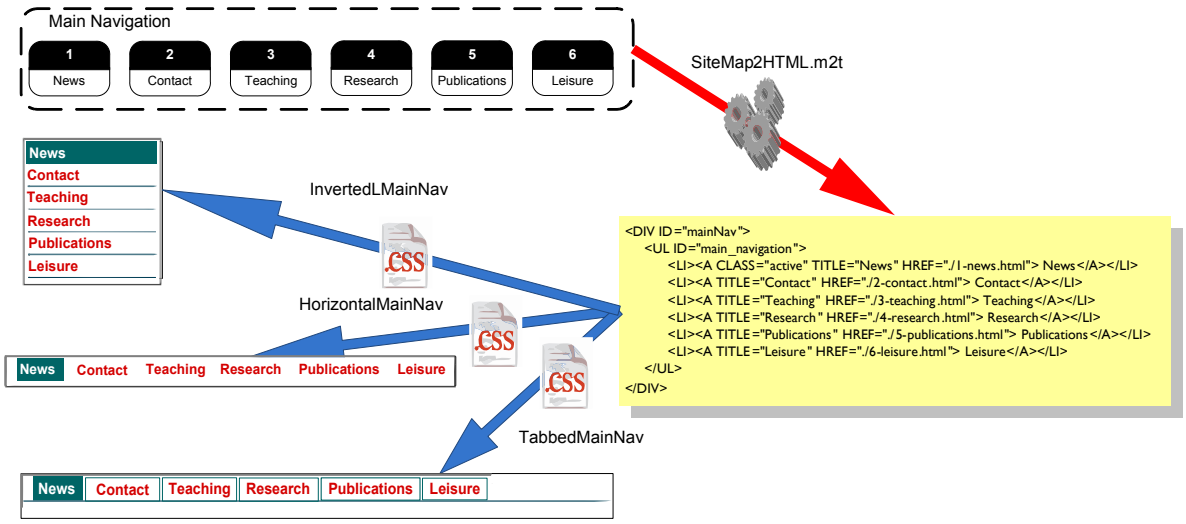


Figura 8.6: Transformando la navegación principal

8.2.3.2 Transformando la navegación local

La navegación local [206] se usa para acceder a los niveles inferiores de la estructura de un sitio, por debajo de la navegación principal. A menudo trabaja en conjunción con la navegación principal, y se puede considerar como una extensión de la misma. Las tres distribuciones más comunes de la navegación local y principal son: L invertida, horizontal y vertical embebida. En el primer caso, la navegación principal normalmente se coloca en la parte superior de la página mientras que la navegación local se representa como una lista vertical de enlaces. En el segundo caso, que es el que se puede apreciar en la figura 8.5, la navegación local se representa como una segunda fila de opciones situada bajo una navegación principal que se representa de forma horizontal. Finalmente, la distribución vertical embebida consiste en mostrar la navegación principal como un menú vertical y entremezclar la navegación local entre los enlaces de la navegación principal, simulando una especie de estructura arbórea.

La figura 8.7 muestra el trozo de código `HTML` generado para la navegación local. La hoja de estilo `CSS` muestra la navegación como una barra horizontal situada por debajo de la navegación principal. En este caso, este trozo de código no se debe insertar en todos los archivos `.html` generados, sino que solamente debe ser incluido en los nodos que pertenecen al área de navegación local (los nodos `5.1`, `5.2`, `5.3` y `5.4` de la figura 11.5), en el nodo padre del área de navegación local, y en los nodos hijos del área de navegación local, si es que existen.

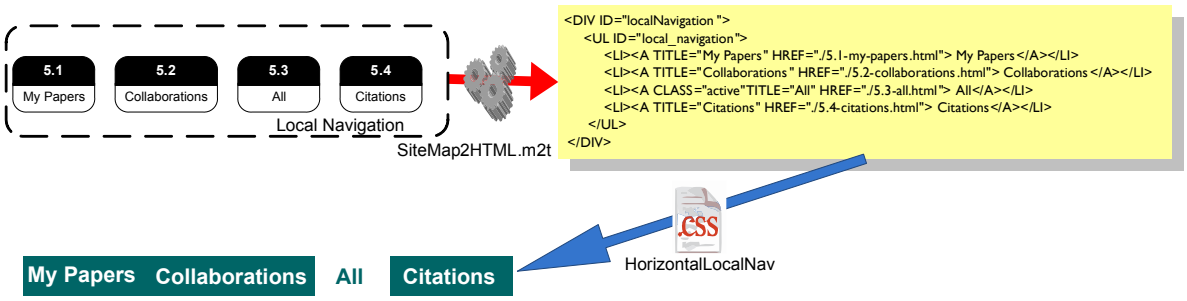


Figura 8.7: Transformando la navegación local

8.2.3.3 Transformando la navegación de utilidad

La navegación de utilidad [367] se usa como suplemento de la navegación principal. Normalmente contiene enlaces a páginas que al diseñador le gustaría tener en todas las páginas del sitio, pero que no tienen el mismo peso visual que la navegación principal. Así, al igual que ocurría con la navegación principal, el trozo de código `HTML` generado para la navegación de utilidad ha de ser incluido en todas las páginas `.html` generadas.

En la figura 8.8 se muestra el trozo de código generado para la navegación de utilidad. El aspecto visual de esta navegación se obtiene aplicando el estilo definido el archivo `utilNav.css`.

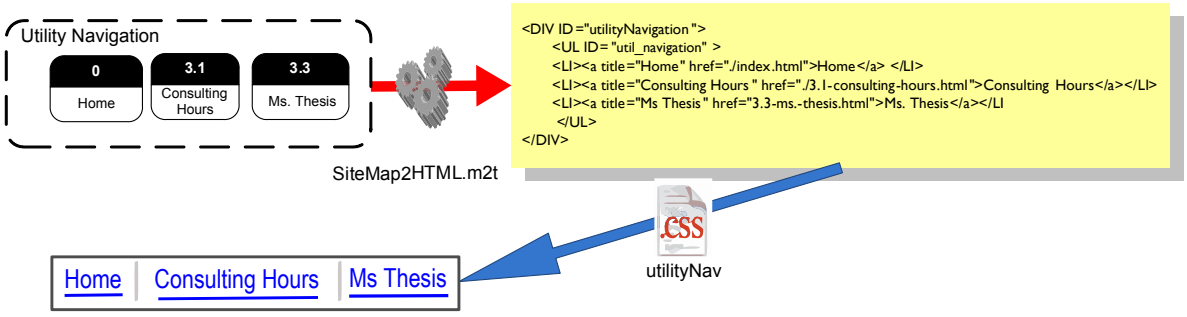


Figura 8.8: Transformando la navegación de utilidad

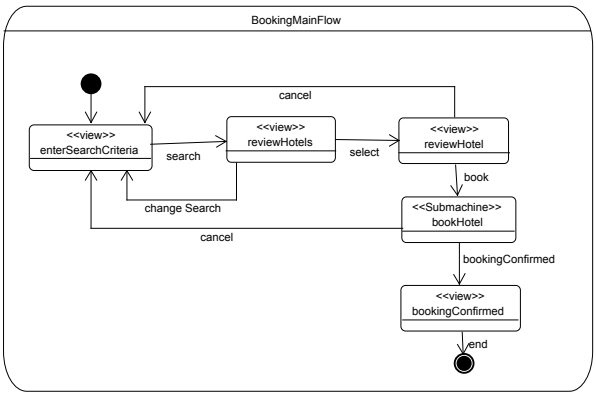
8.3 Flujos de navegación

Un flujo de navegación es un modelo de la interacción que un usuario puede tener con una aplicación web [391]. Un flujo reacciona a eventos de usuario para conducir hasta la terminación del mismo. Según [391] el usar diagramas de estado como base para definir los flujos de página en aplicaciones web es mucho más natural que las soluciones

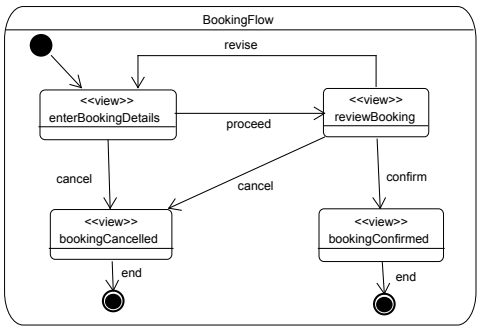
centradas en peticiones, especialmente en los casos de uso más complejos en los que el usuario tiene que dar una serie de pasos bien definidos para completar un proceso de negocio.

En general, por la forma en que se diseñó la Web, como un entorno cliente-servidor en el que no se mantiene el estado, los usuarios pueden navegar libremente por distintas páginas. Sin embargo, en muchas aplicaciones web modernas se requiere algo más de control sobre lo que está haciendo el usuario, restringiendo la libertad del mismo para hacer determinadas operaciones y obligándolo a seguir determinados pasos hasta completar un proceso. Un ejemplo de este tipo de navegación a la que se conoce como *navegación controlada* es el proceso que el usuario tiene que seguir una vez que ha decidido formalizar una compra en una tienda de comercio electrónico como `Amazon`. Normalmente, se eliminan el resto de enlaces del sitio para que la única operación que el usuario pueda hacer sea avanzar hasta el siguiente paso del proceso y no se le permite realizar ninguna otra operación hasta que, o bien complete el proceso, o bien lo cancele.

En la figura 8.9 se muestra un ejemplo de flujo para realizar la reserva de una habitación en un hotel. En la parte izquierda (figura 8.9(a)) se muestra el flujo principal, mientras que en la parte derecha (figura 8.9(b)) se muestra el subflujo que se dedica a realizar la reserva. Normalmente el flujo principal se diseña de forma que al usuario se le permite realizar otras operaciones, sin embargo, en el momento en que el usuario entra en el estado `bookHotel`, la navegación pasa a ser controlada. El usuario notará visualmente este efecto gracias a que al entrar en el subflujo de la (figura 8.9) desaparecerán de la pantalla todos los enlaces, excepto aquéllos que permitan avanzar en el subflujo o cancelar la operación.



(a) Flujo principal para realizar la reserva de habitaciones de un hotel.



(b) Subflujo de reserva

Figura 8.9: Diálogo para realizar la reserva de una habitación de un hotel

En las siguientes secciones se muestra la definición de la sintaxis abstracta de un lenguaje basado en diagramas de estado para tratar con flujos web, y una sintaxis concreta para este mismo lenguaje.

8.3.1 Metamodelo

En esta sección se describe el metamodelo utilizado para definir el comportamiento de la navegación. Este metamodelo está basado en la propuesta de Winckler [411]. En concreto, es una adaptación del `DTD` publicado en [412], aunque se ha introducido algunas diferencias:

1. Consideramos que los estados básicos representan trozos de información que se les muestra al usuario (como símil podríamos pensar en las distintas secciones `<div>` que componen una página web). Así, una página web puede verse como un estado compuesto que tiene varias regiones. El número de regiones dependerá de los trozos de información que se le muestran al usuario. En este sentido, se puede decir que nuestros estados son de granularidad más fina que [411], pero de grano más grueso que los propuestos en [342], una propuesta para testear aplicaciones web en la que los estados representan elementos atómicos de las páginas web, tales como botones, elementos de formularios o imágenes.

2. Se ha introducido un nuevo tipo de estado llamado `SubflowState` como una forma de estructurar mejor los modelos.

3. Se ha introducido un nuevo tipo de estado, llamado `FrontEndView`, como subtipo de `CompositeState` que representa la vista de una página web que se le muestra al usuario.

4. Se distingue entre navegación controlada y no controlada gracias a la propiedad `controlled` que se ha añadido a la metaclase `WebFlow`.

5. Se ha añadido una propiedad a las transiciones que indica si esa transición producirá múltiples enlaces en la página destino.

El metamodelo que define los constructores de nuestra propuesta y sus relaciones se puede ver en la figura 8.10. La raíz del metamodelo es la metaclase `WebFlow`. La mayoría de las propiedades de esta clase (`project`, `author`, `date` y `version`) se han añadido con el propósito de documentar el flujo web, por lo que todas tienen cardinalidad `0..1`, es decir, son opcionales. La única propiedad significativa con respecto al comportamiento del flujo es `controlled`, que indica si el flujo es de navegación controlada o no. Que el flujo sea de navegación controlada implica que, en algunas ocasiones, el resto de *concerns* de navegación desaparecerán de la pantalla, dejando sólo los enlaces correspondientes al propio flujo. El valor por defecto para esta propiedad es `false`. Si un flujo tiene partes en las que la navegación es controlada y partes en las que no, se modelará como un flujo con un estado de tipo subflujo. Cuando se detalle el subflujo, se indicará que es de navegación controlada asignando a su propiedad `controlled` el valor `true`.

Un flujo web está compuesto por cero o más transiciones (`Transition`) y exactamente un nodo de tipo (`CompositeState`), que hace de contenedor del flujo. El resto de

estados del flujo estarán incluidos en este estado compuesto que hace de contenedor, teniendo en cuenta que al menos uno de estos estados tiene que ser de tipo `FrontEndView`, lo que se especifica mediante una restricción `OCL`, que se verá más adelante.

Una transición (`Transition`) está relacionada con un nodo (`Node`) mediante la relación llamada `target`, que indica cuál es el nodo destino al que se dirige la misma. Una transición también está relacionada con la metaclase `State` a través de la relación `source`, que indica cuál es el estado del que parte la transición. Las relaciones opuestas a `target` y `source` son `incomingTransitions` y `outgoingTransitions`, respectivamente. Las transiciones pueden tener asociada una acción que se ejecutará cuando se dispare la transición. Esto se ha modelado mediante la relación `action`. Las propiedades de una transición son las siguientes: `id`, que representa un identificador único; `trigger`, que es el nombre del evento que lanza la transición; `guard`, que representa una condición que se ha de cumplir para que se dispare la transición; `type`, que indica el tipo de transición; y, finalmente, `multiple`, que indica si una transición produce varios enlaces en la vista destino o no.

El tipo de transición se ha modelado mediante la enumeración `TransitionKind` cuyos valores son: `USER`, el valor por defecto, que modela una transición que se dispara a causa de alguna acción del usuario; `SYSTEM`, que representa a una transición disparada por algún evento del sistema, como por ejemplo la ejecución de un método; y, finalmente, `COMPLETION`, que es un tipo de transición que se dispara cuando el sistema cambia después de un tiempo.

Los nodos (`Node`) pueden ser estados (`State`) y pseudoestados (`Pseudostate`). Una de las diferencias más importante entre estados y pseudoestados es que estos últimos no tienen transiciones salientes. Los estados se clasifican en básicos (`BasicState`), compuestos (`CompositeState`) y subflujos (`SubflowState`). Los estados básicos representan trozos de información que se le muestran al usuario. Los estados compuestos modelan una página o una sección de una página compleja, mientras que los subflujos son una forma de simplificar los modelos mediante jerarquías de estados. Las páginas se modelan mediante `FrontEndView`, un tipo de estado compuesto.

Los estados básicos pueden ser de cuatro tipos: `STATIC`, `DYNAMIC`, `TRANSIENT` y `EXTERNAL`. Los posibles valores para los tipos de estado se han modelado mediante la enumeración `StateKind`. `STATIC` representa un estado cuyo contenido asociado es fijo, se puede decir, por tanto que el contenido asociado es estático. `DYNAMIC` representa un estado cuyo contenedor asociado contiene enlaces cuyo destino es calculado automáticamente por el sistema en tiempo de ejecución. Los estados de tipo `TRANSIENT` son estados que no tienen asociados ningún tipo de contenedor y, como consecuencia, no tienen representación visual. `EXTERNAL` representa a un estado con información a la que se puede acceder pero que está fuera del ámbito del sitio web. Normalmente se usa para modelar enlaces a sitios externos. Además, de la propiedad `type`, los estados básicos también tienen la propiedad `file` que guarda o bien la ruta de un fichero local, o bien la `URL` de un sitio externo al que apunta el estado, en el caso de que el estado sea de tipo `EXTERNAL`.

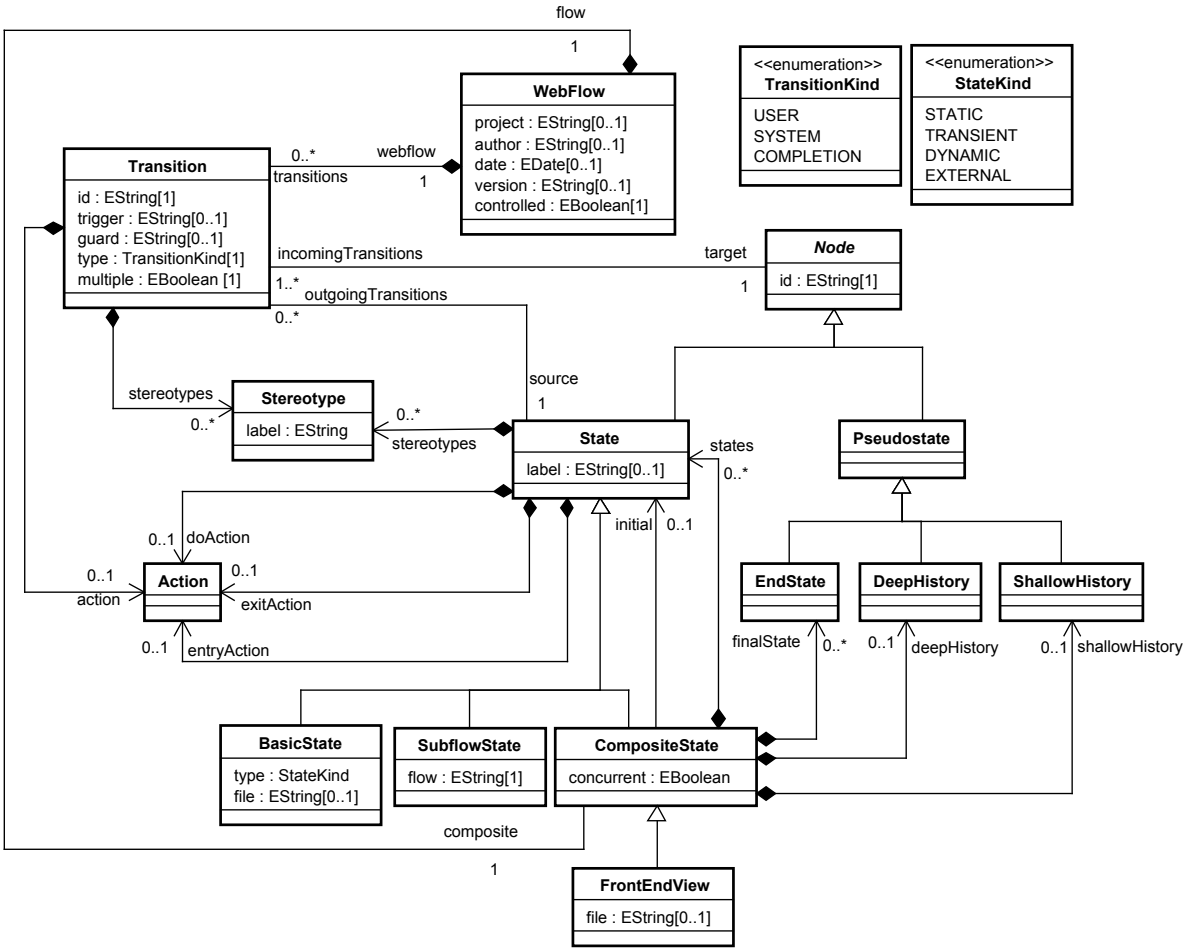


Figura 8.10: Un metamodelo para un lenguaje para expresar flujos de navegación

Un estado de tipo `CompositeState` tiene una propiedad que indica si los estados que contiene están en regiones concurrentes o no. Las regiones concurrentes representan áreas que se muestran al usuario de forma concurrente, tal y como *frames* o regiones de tipo `<div>`. Además, como hace de contenedor, un estado de tipo `CompositeState` puede contener de cero a varios estados, cero o un estado final, cero o un pseudoestado de tipo `DeepHistory` y cero o un pseudoestado de tipo `ShallowHistory`. Si el estado compuesto contiene uno o más estados, entonces uno de ellos tiene que ser marcado como estado inicial. Este requisito se expresa mediante una restricción `OCL`, tal y como se verá más adelante.

Un estado de tipo `SubflowState` tiene una propiedad `flow` que representa el flujo que se dispara al entrar en este estado. Finalmente, todos los estados tienen una etiqueta, que se almacena en la propiedad `label`, y un identificador, que será único para todos los nodos.

El historial de visitas que se implementa en la mayoría de los navegadores se modela mediante los pseudoestados `DeepHistory` y `ShallowHistory`. Mientras que `DeepHistory` especifica los últimos pasos de navegación (similar al botón `Historial` de un navegador), `ShallowHistory` modela el hecho de volver atrás al estado anterior (semejante al botón `Atrás`).

Una acción (`Action`) representa una funcionalidad que se ha de ejecutar en algún punto de la secuencia de navegación. Los puntos en los que se puede ejecutar en este caso son: antes de entrar en el estado, al salir del estado y dentro del estado. Esto se ha modelado mediante las relaciones de composición que parten de `State` y se dirigen hacia `Action` llamadas `entryAction`, `exitAction` y `doAction`, respectivamente. En una transición también se puede ejecutar una acción, lo que se modela mediante la relación `action` entre `Transition` y `Action`.

Además de estos tipos de estados y transiciones predefinidos, éstos se pueden ampliar mediante la definición de estereotipos, hecho modelado mediante la metaclase `Stereotype`.

8.3.1.1 Restricciones OCL

El metamodelo definido en la figura 8.10 se acompaña con una serie de restricciones especificadas en `OCL`. Estas restricciones se han implementado usando el *plugin* de Eclipse para `OCL` incluido dentro del proyecto de modelado Eclipse `EMP`. Algunas de estas restricciones se han introducido para mantener simplificado al máximo el metamodelo. A continuación, se especifican éstas, agrupadas por el contexto al que se aplican.

Contexto WebFlow. Dentro del contexto de la metaclase `WebFlow` se han definido las siguientes restricciones (listado 8.5):

Listado 8.5: Restricciones OCL asociadas al contexto WebFlow

```
1 context WebFlow
2 inv: transitions->isUnique(id)
3 inv: Node.allInstances()->isUnique(id)
4 inv: composite.states->select(oclIsTypeOf(FrontEndView))->size()>=1
```

1. (Línea 2) El identificador de las transiciones debe ser único.
2. (Línea 3) El identificador de los nodos debe ser único.
3. (Línea 4) El estado compuesto que hace de contenedor ha de tener al menos un estado de tipo `FrontEndView`.

Contexto CompositeState. Dentro del contexto de la metaclase `CompositeState` se han definido las siguientes restricciones (listado 8.6):

Listado 8.6: Restricciones OCL asociadas al contexto CompositeState

```
1 context CompositeState
2 inv: states->size()>=1 implies not initial.oclIsUndefined()
```

1. (Línea 2) Si el estado compuesto contiene estados, entonces debe haber un estado inicial.

8.3.2 Una sintaxis concreta

Hasta ahora se ha definido la sintaxis abstracta para definir flujos web. La sintaxis concreta propuesta se ha definido basándose en la sintaxis concreta de `UML` para los diagramas de estado. Como el autor de StateWebCharts, la propuesta en la que se basa este lenguaje ya había definido una sintaxis concreta, se ha decidido, para algunos elementos, proponer sintaxis alternativas, sobre todo cuando la original no coincide exactamente con la sintaxis concreta de `UML`. Así, en la figura 8.11 se muestran dos sintaxis alternativas para los distintos tipos de estados. Mientras que en la fila superior se ha representado la sintaxis que proporcionó el autor original de la propuesta, en la fila inferior se indica la sintaxis de nuestra propuesta. Básicamente, la sintaxis propuesta para los estados consiste en indicar, a modo de estereotipo, el tipo de estado que se representa. Así, se han definido los estados `<<DynamicState>>`, `<<TransientState>>` y `<<ExternalState>>`. Cuando no se indica ningún estereotipo el estado es de tipo `Static State` o `Composite State`. La diferencia entre ambos reside en que en los estados compuestos hay una región en la que se incluirán los estados pertenecientes al estado compuesto. La sintaxis concreta representada en la parte inferior es la misma que proporciona `UML` para estos tipos de estado. Finalmente, en todos los estados representados en la figura 8.11 se representa su identificador (`ID`) seguido de su nombre (`name`), que se toma de la propiedad `label` del metamodelo de la figura 8.10.

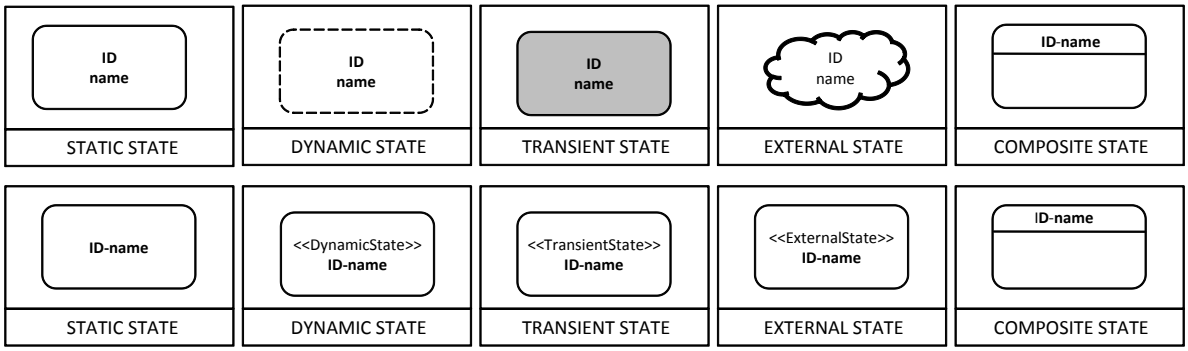


Figura 8.11: Sintaxis concretas alternativas para los estados pertenecientes a la propuesta original

Siguiendo la misma filosofía, la figura 8.12 muestra la sintaxis concreta de los estados que se han añadido a la propuesta original. Como `FrontEndView` es un estado de tipo compuesto, se representa igual que éstos, pero con el estereotipo `view`. Finalmente, un estado de tipo subflujo estará estereotipado con `Subflow`.

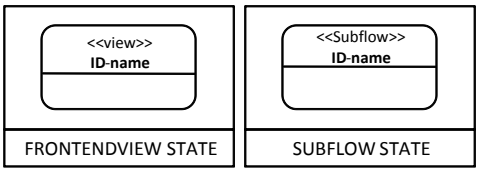


Figura 8.12: Sintaxis concretas para los estados añadidos a la propuesta original

Los pseudoestados se muestran en la figura 8.13. En este caso no se han definido sintaxis concretas alternativas puesto que la sintaxis propuesta por los autores de la propuesta original coincide con la sintaxis de `UML`. Como puede comprobarse en la figura, se ha definido la sintaxis concreta para cuatro pseudoestados (`End State`, `Deep History`, `Shallow History` e `Initial State`). Sin embargo, si nos fijamos en el metamodelo de la figura 8.10, comprobaremos que solamente hay tres clases hijas de la metaclase abstracta `PseudoState` (`EndState`, `DeepHistory` y `ShallowHistory`). El cuarto pseudoestado tiene un comportamiento distinto a los anteriores y representa a la relación `initial` entre `CompositeState` y `State`. También hay que destacar que la única diferencia entre la sintaxis concreta de los dos pseudoestados de tipo historial es que en el historial en profundidad el símbolo que aparece en el interior del círculo es una hache mayúscula (`H`) seguida de un asterisco (`*`) representado como superíndice, mientras que el historial inmediatamente anterior solamente contiene una hache mayúscula (`H`).

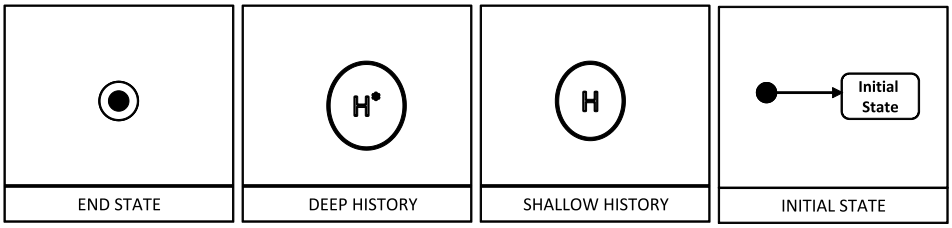


Figura 8.13: Sintaxis concreta para los pseudoestados

Finalmente, los tipos de transiciones se recogen en la figura 8.14. En este caso tampoco hay diferencia entre la propuesta original y la sintaxis que se propone en este capítulo. Como se puede comprobar, se han definido tres sintaxis para los tres tipos de transiciones (`USER TRANSITION`, `SYSTEM TRANSITION` y `COMPLETION TRANSITION`) y una sintaxis basada en la de transición de usuario para aquellas transiciones de usuario

múltiples. Las transiciones que disparan los eventos generados por los usuarios al interactuar con la aplicación se representan de la misma forma que cualquier transición en UML, es decir, una línea que parte de un estado origen (`Source State`) y termina en un estado destino (`Target State`). El extremo de la línea que linda con el estado destino tiene una punta de flecha apuntando hacia dicho estado destino. Si la transición de usuario puede producir varios enlaces en el estado destino, entonces se representa mediante un asterisco asociado a la propia transición. Las transiciones del sistema y las que se autocompletan tienen una representación similar, la única diferencia es que en estas dos la línea que une los estados se dibuja pespunteada. Por último, a cada transición se le asocia su identificador (`id`), el evento que la dispara (`userEvent` en el caso de transiciones disparadas por eventos de usuario y `systemEvent` en el de transiciones disparadas por eventos del sistema), la condición (`condition`) que se ha de cumplir para lanzar la transición, y la acción a ejecutar (`action`).

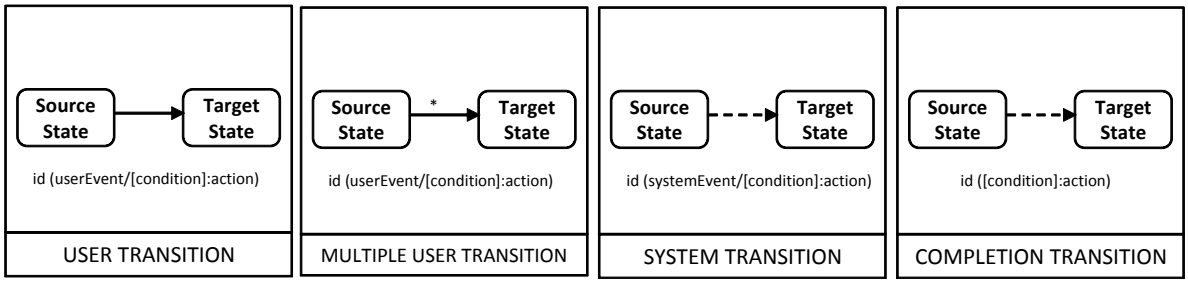


Figura 8.14: Sintaxis concreta para las transiciones

8.4 Sumario

Dentro del marco de MWACSL este capítulo se ha centrado en estudiar en profundidad el aspecto de navegación a nivel independiente plataforma. Como la navegación es un dominio complejo, se descompone en varios *concerns*, de forma que sea más fácil trabajar con él. Este capítulo se centra en dos de esos *concerns*, la navegación estructural y de utilidad y los flujos de navegación web, que pueden categorizarse como *concerns* navegacionales "puros" y de tarea, respectivamente.

Una primera versión de la sintaxis abstracta del lenguaje específico de dominio propuesto para tratar con la navegación estructural y de utilidad, se publicó en [328], así como las transformaciones de modelo a texto que sirven como prueba de conceptos del propio metamodelo. Este metamodelo también se ha introducido en [329], pero esta vez enfocado en transformaciones modelo a modelo. Hay que resaltar que el lenguaje presentado en estos artículos ha evolucionado hasta la versión que se ha presentado en este capítulo. Además, en el capítulo también se ha presentado la sintaxis abstracta y

concreta de un lenguaje para tratar con *concerns* navegacionales de tarea basados en *statecharts*.

A good idea is about ten percent and implementation, hard work, and luck is 90 percent.

Guy Kawasaki

9

El Reflejo de la Navegación en Plataformas Concretas

En este capítulo se presentan, en el marco de `MWACSL`*, los lenguajes de modelado definidos a nivel específico de plataforma usados para implementar los* concerns *navegacionales separados en el capítulo anterior. La sección 9.2 se centra en la especificación de un lenguaje de modelado para* `HTML` *y en cómo la navegación estructural y de utilidad se reflejan en esta plataforma. En la sección 9.3 se especifica el lenguaje de modelado para Spring Web Flow, una plataforma para especificar la navegación controlada. La sección 9.4 describe el lenguaje de modelado de* `JSP` *que se ha usado en la tesis. Este último metamodelo no es una contribución de los autores, puesto que se ha definido dentro del proyecto Modisco [125], aunque es necesario para entender algunas de las transformaciones descritas en capítulos posteriores. Finalmente, en la sección 9.5, se concluye el capítulo.*

9.1 Introducción

Este capítulo se centra en los lenguajes de modelado definidos a nivel específico de plataforma usados para implementar los distintos *concerns* de navegación. Hay que tener en cuenta que la separación que se ha conseguido a nivel independiente de plataforma, no siempre se puede mantener a nivel específico de plataforma. Así, hasta donde alcanza nuestro conocimiento, no existe ninguna plataforma concreta que permita definir de forma aislada la navegación estructural y de utilidad. Por lo tanto, este tipo

de navegación se ha implementado en plataformas comunes en las que no se mantienen separados: HTML y JSP.

Los flujos de navegación controlada se han implementado en Spring Web Flow (SWF), una plataforma que se ha escogido principalmente por tres motivos: en primer lugar, porque aplica el principio de separación de conceptos y no mezcla los flujos de navegación con ningún otro *concern*, lo que permite, dentro de MWACSL, centrarse solamente en la especificación de ese concepto, y mantener una separación limpia desde el nivel independiente de plataforma hasta el dependiente de plataforma; en segundo lugar, porque maneja un conjunto de conceptos relativamente pequeños, con lo que la tarea de modelado se hace más liviana; y, en tercer lugar, porque permite definir los flujos tanto con clases Java como con archivos XML, haciendo en este último caso más simple la especificación de las transformaciones modelo a texto.

Los flujos de navegación también se han implementado en JSP, ya que este *concern* se esparce por distintos puntos de un programa. A la vez que el flujo es especificado de forma separada en un archivo de Spring Web Flow, tiene su reflejo los archivos JSP que implementan la vista asociada a los estados que se definen en el flujo.

Para cada una de estas plataformas se ha definido un metamodelo que especifica los conceptos que se manejan y la relación entre ellos. La estrategia para definir estos metamodelos ha sido distinta, en el caso del metamodelo de SWF, se ha estudiado la plataforma y se ha definido desde cero. En el de HTML, se utiliza la infraestructura que proporciona EMF para no tener que implementarlo desde cero, con lo que solamente hay que realizar pequeños retoques al metamodelo Ecore obtenido a partir del XML Schema proporcionado por el consorcio W3C. Finalmente, el metamodelo de JSP se ha reutilizado totalmente del proyecto Modisco [125].

9.2 Navegación estructural y de utilidad en una plataforma concreta

9.2.1 Metamodelo de HTML

El metamodelo de HTML no se ha definido de forma manual, sino que se ha obtenido de forma casi automática utilizando el componente del *plugin* de Eclipse EMF para importar modelos XSD, tal y como se menciona en [156]. Este componente toma como entrada un archivo XML Schema Definition (XSD), y genera un archivo .ecore. La entrada se ha tomado directamente de la especificación que el consorcio W3C tiene publicada en [397]. La figura 9.1 muestra un extracto de este metamodelo representado con el editor en forma de árbol de EMF. La metaclase DocumentRoot es la raíz del metamodelo y muchas de las metaclases del mismo se corresponden con etiquetas HTML. Así, por ejemplo, la etiqueta <a> tiene asociada la metaclase AType (octava metaclase del árbol de la figura 9.1).

Anteriormente se ha comentado que el metamodelo se ha obtenido de forma casi

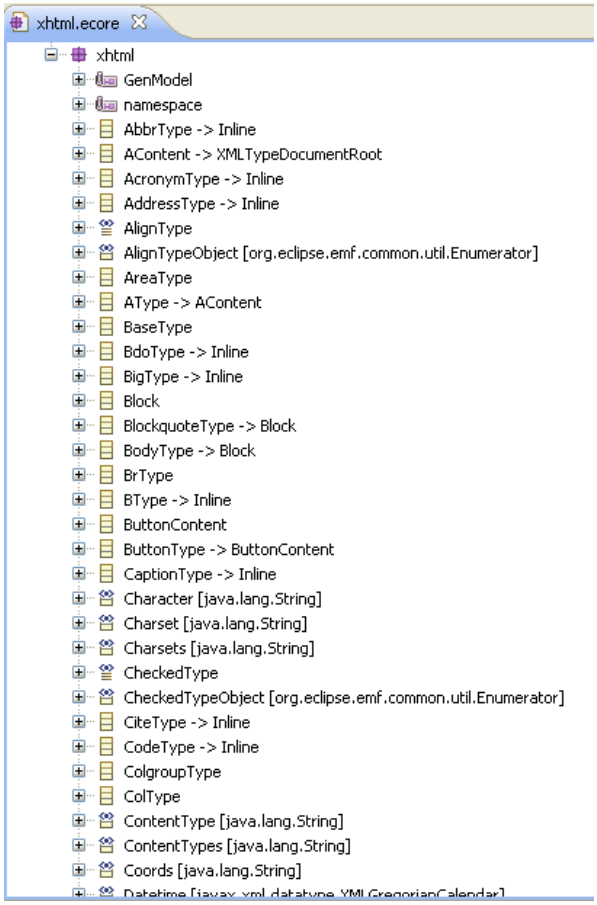


Figura 9.1: Extracto del metamodelo de HTML

automática, y es que al archivo `.ecore` obtenido de la importación se le han tenido que hacer algunas modificaciones menores. Estas modificaciones se deben al hecho de que no hay manera de declarar elementos textuales entre etiquetas `HTML` tales como `<li>` o `<p>` mediante el metamodelo importado [156]. Es decir, no se podría modelar algo como `<p>Párrafo</p>`. Para solucionar este problema se ha añadido una generalización entre la metaclase `XMLTypeDocumentRoot` y las metaclases `AContent`, `Flow`, `Inline` y `TitleType`. Como consecuencia de esta modificación en la jerarquía de clases, se ha de eliminar el atributo llamado `mixed` de estas cuatro metaclases, ya que ahora lo heredan de su padre (la metaclase `XMLTypeDocumentRoot`).

La última modificación está relacionada con el hecho de facilitar las transformaciones modelo a modelo, y está relacionada con la metaclase `AType`. El tipo del atributo `class` de esta clase se ha cambiado de `NMTOKENS` a `NMTOKEN` con multiplicidad [0..n].

9.2.2 Metamodelo de Website

El metamodelo denominado `website` se ha definido como una forma de agrupar los diferentes archivos `HTML` que componen el sitio web. Por lo tanto, depende del metamodelo de `HTML` definido en la sección anterior. A la vez, el propio metamodelo de `HTML` depende de `XMLType` y de `XMLNamespace`. Las relaciones entre estos paquetes se muestran en la figura 9.2.

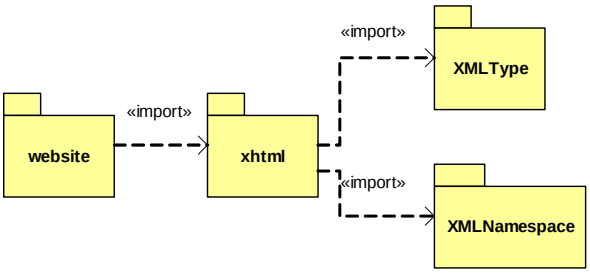


Figura 9.2: Relaciones de dependencia entre metamodelos

La raíz del metamodelo `website` es la metaclase llamada `WebSite`, que solamente tiene dos propiedades `id`, que representa un identificador único para el sitio web, y `name`, que contendrá el nombre del sitio web. El sitio web estará compuesto por uno o más ficheros `HTML` (metaclase `HTMLFile`). Cada archivo `HTML` se almacena en un directorio concreto (propiedad `dir`) y tiene un nombre (`filename`). Además, el contenido del archivo se modela mediante la relación `content` con la metaclase `DocumentRoot`, que es la raíz del metamodelo `HTML`, descrito en el apartado anterior. Este hecho se ha representado estereotipando con `<<from xhtml>>` la metaclase `DocumentRoot` (figura 9.3).

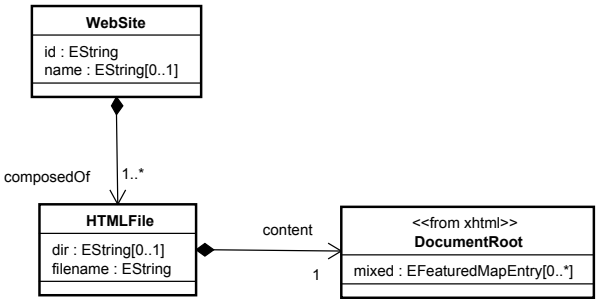


Figura 9.3: Metamodelo de website

9.2.3 De sitemap a website y HTML mediante transformaciones M2M

La transformación toma como entrada un modelo que sea conforme al metamodelo de `sitemap` definido en la sección 8.2.1, y produce un modelo conforme al metamodelo

de `website` descrito en la sección 9.2.2 del capítulo 9. La idea principal que subyace tras la transformación es que, objetivamente, la navegación de utilidad y estructural son básicamente una lista de enlaces a varias páginas del sitio web [149]. Como el foco de la transformación es la navegación, lo que se va a generar es un esqueleto del sitio conteniendo exclusivamente enlaces que van a representar la navegación de utilidad y estructural, sin llegar a generar todo el contenido de las páginas.

Una página generada por la transformación estará estructurada en diferentes áreas especificadas mediante etiquetas `<div>`. Un esquema de estas áreas se muestra en la figura 8.4. En esta figura los rectángulos representan las diferentes áreas definidas mediante `<div>`. Posteriormente, aplicando hojas de estilo, se podrá obtener o bien la distribución en L invertida de la figura 8.4(a) o la distribución horizontal de la figura 8.4(b). Lo que es interesante resaltar en este caso es que, independientemente de la distribución, las áreas generadas serán las mismas: `container`, `header`, `utilityNavigation`, `mainNavigation`, `localNavigation`, `mainContent` y `footer`. El área `<div>` con atributo `id` igual a `container` se situará justo por debajo de la etiqueta `<body>`. Además, debajo de `container` se generarán las seis divisiones con los atributos `id` tal y como se muestra en la parte derecha de la figura 9.4. Esta figura muestra cuál sería el resultado de la transformación de un nodo genérico.

En la parte izquierda de la figura aparece una instancia de la metaclase `Node` (perteneciente al metamodelo `sitemap`, el origen de la transformación). En la parte derecha se muestran el conjunto de metaclases pertenecientes al metamodelo `website` (metamodelo destino) que se obtendrán a partir de este nodo. Como puede comprobarse, por cada instancia de `Node` se generará una instancia de la metaclase `HTMLFile`, que estará relacionada con una instancia de `DocumentRoot`, que representa la raíz del archivo `HTML`. El resto de metaclases de la parte derecha representan etiquetas `HTML`. Así, por ejemplo, la instancia de `HTMLType` representa la etiqueta `<hmtl>`. La figura también muestra cómo se generan diferentes instancias de la metaclase `DivType`, una con `id` igual a `container`, dependiendo de una instancia de `BodyType`, y el resto, dependiendo de ésta. Nótese también que solamente aquellas instancias de `DivType` relacionadas con la navegación (las que tienen como `id` a `utilityNavigation`, `mainNavigation` y `localNavigation`) tienen por debajo instancias de otras metaclases. En concreto, cada una de ellas está relacionada con una instancia de `UlType`, ya que la navegación se va representar como una lista `HTML`. Cada instancia de `UlType` tendrá asociadas un número variable de instancias de `LiType`. Este número dependerá de la navegación que se representa. Así por ejemplo, si el área de navegación de utilidad contiene tres nodos, entonces bajo la instancia de `UlType` asociada a la navegación de utilidad se generan tres instancias de `LiType`, representando tres etiquetas `<li>`. Además, cada una de estas instancias de `LiType` estará asociada con una instancia de `AType`, que representará la etiqueta `<a>` que contendrá la ruta para llegar al fichero `HTML` que representa el nodo de la navegación de utilidad.

La transformación se ha implementado mediante la implementación de QVT Operational Mappings [99] incluida en el subproyecto M2M del Eclipse Modeling Project.

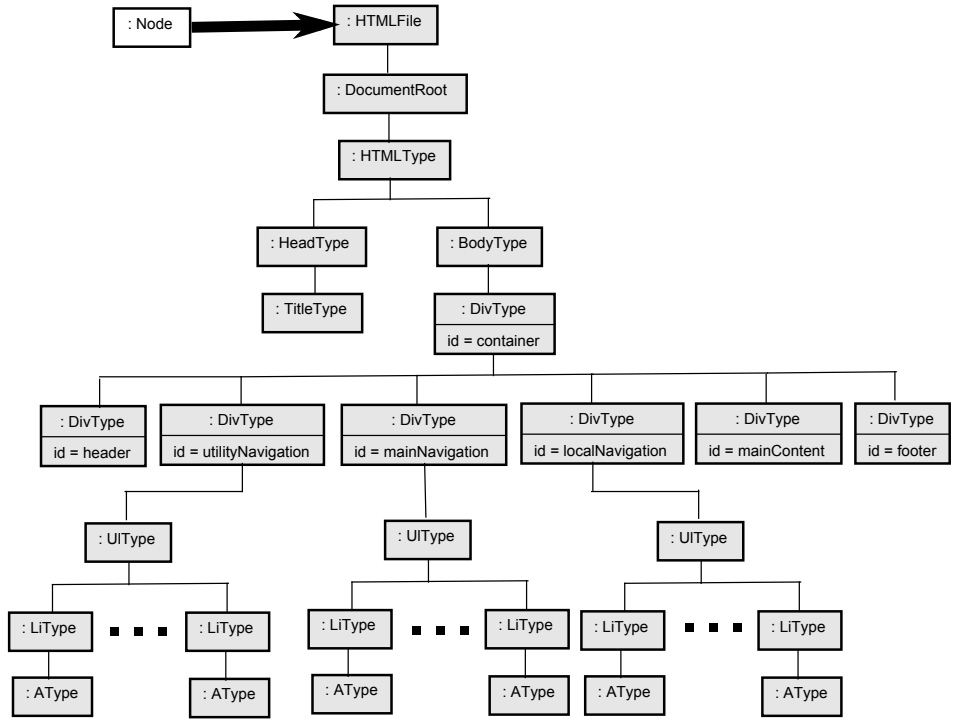


Figura 9.4: Resultado de la transformación aplicada a un nodo

La tabla 9.1 muestra un resumen de las operaciones definidas en la transformación. El código de la misma puede verse en el sitio web de `MWACSL` (`http://www.lsi.us.es/~reinaqu/org.mwacsl/html/`).

Se puede observar que hasta ahora no se ha dado ningún detalle acerca de la presentación. Esta información se incluye en las transformaciones mediante cuatro propiedades de configuración llamadas `cssContainer`, `cssUtilityNav`, `cssMainNav` y `cssLocalNav`. Cada propiedad guarda la `URI` del archivo de hoja de estilo que se aplicará para definir el aspecto visual de cada una de las áreas. Así, la propiedad `cssMainNav` contendrá la `URI` del fichero `CSS` que tendrá el aspecto de la navegación principal.

9.3 Una plataforma para definir la navegación controlada: Spring Web Flow (SWF)

`SWF` [89] es un *framework*, situado por encima de *Spring Framework* [201], que permite la definición y representación de flujos de interfaces de usuario en las aplicaciones web de una forma clara y simple. Un *flujo* define un diálogo con el usuario, de tal forma que las respuestas a los eventos producidos por el usuario durante el diálogo van guiando la ejecución del código de la aplicación.

Tipo Operación	Regla

Mapping	`main()`
	`Sitemap::toWebSite() : webSite::WebSite`
	`siteMap::Node::toXHTMLFile(): webSite::HTMLFile`
	`siteMap::Node::toHTMLDocumentRoot(): xhtml::DocumentRoot`
	`siteMap::Node::toDivHeader(): xhtml::DivType`
	`siteMap::Node::toDivUtilityNavigation(): xhtml::DivType`
	`siteMap::Node::toDivMainNavigation(): xhtml::DivType`
	`siteMap::Node::toDivLocalNavigation(): xhtml::DivType`
	`siteMap::Node::toDivMainContent(): xhtml::DivType`
	`siteMap::Node::toDivFooter(): xhtml::DivType`

Constructor	`xhtml::LinkType::LinkType(in hrefStylesheet:String)` `xhtml::LiType::LiType(current: siteMap::Node, iter:siteMap::Node)`

Query	`siteMap::Node::isDescendantOf(node:siteMap::Node): Boolean`
	`siteMap::Node::getLocalNavigationArea():siteMap::Area`
	`siteMap::Node::affectedByLocalNav(in area: siteMap::Area):Boolean`
	`getMainNavigationArea(): siteMap::Area`
	`getUtilNavigationArea(): siteMap::Area`
	`siteMap::Sitemap::getNodes(): Set(siteMap::Node)`
	`getAreaNodes(in area: siteMap::Area): Set(siteMap::Node)`
	`getNodeDescendents(in nodo: siteMap::Node): Set(siteMap::Node)`
	`String::toFilenameConvention():String`
	`siteMap::Node::fullFileName():String`
	`guardText(in text: String) : String`
	`guardTexts(in text: String, in text2: String) : String`

Tabla 9.1: Resumen de operaciones definidas en `sitemap2website`

En `SWF` los flujos se pueden definir de forma declarativa gracias a un `DSL`. `SWF` permite trabajar con este lenguaje tanto con archivos `XML` como a través de un conjunto de clases Java. Un flujo está formado por un conjunto de uno o más estados. Cada *estado* define un paso dentro del flujo del ejecución del diálogo, de tal forma que al entrar en cualquier estado se ejecuta un comportamiento asociado al mismo. Este comportamiento dependerá tanto de la configuración como del tipo de estado en el que se entra. Un *evento* causará una transición de estados, así, por ejemplo, cuando el usuario pulsa un botón, se lanzará un evento de tipo *submit*. Este evento tendrá una correspondencia con una transición que provocará un cambio de estado, por ejemplo, al estado `PedidoCompletado`. Así, en `SWF` se definen cinco tipos de estados:

ViewState Es el estado que permite al usuario (o a un cliente externo) participar en el flujo. Cuando el flujo entra en un estado de este tipo, se detiene, y se devuelve el control al sistema que lo invocó, y que será el encargado de mostrar una interfaz al usuario. Después, el usuario, mediante la interacción con la interfaz, producirá un evento que provocará la reanudación de la ejecución del flujo.

ActionState Este tipo de estado es el utilizado para ejecutar código con la lógica de negocio de la aplicación. El resultado de la ejecución de este código decidirá cuál será el siguiente estado en el que entrará el flujo. La *acción* es la abstracción que se utiliza para encapsular la ejecución de una unidad lógica del código de la aplicación. Cuando el flujo entra en un estado de tipo `ActionState`, se van ejecutando en orden un conjunto de acciones. Si el resultado de la salida de la

primera acción ejecutada concuerda con una transición, entonces se ejecuta la transición. En caso contrario, se ejecutará la siguiente acción de la lista.

DecisionState Este estado permite cambiar la ruta del flujo dependiendo de una decisión. Cuando el flujo entra en un estado de tipo `DecisionState`, se evalúan un conjunto de expresiones booleanas. El resultado de la evaluación decidirá cuál será el siguiente estado en el que entrar.

SubflowState Con este tipo de estado se permite la reutilización de un flujo ya existente, de tal forma que, cuando se entra en un estado de esta clase, se ejecuta un nuevo flujo como subflujo del anterior. Al subflujo se le pueden pasar parámetros de entrada y puede devolver una serie de valores de retorno.

EndState Es el estado que indica la terminación del flujo. Así que, cuando un flujo entra en un estado de tipo `Final`, termina su ejecución. Si el flujo estaba actuando como subflujo, la sesión termina y continúa la ejecución del flujo padre.

9.3.1 El metamodelo de SWF

En esta sección se describe el metamodelo utilizado para definir los flujos de Spring Web Flow a nivel específico de plataforma. La implementación del metamodelo se ha realizado utilizando `EMF` [49, 311]. Aunque el estándar de definición de metamodelos definido por la `OMG` es `MOF` [286] se puede definir una equivalencia entre ambos lenguajes de metamodelado [255, 138, 92], de tal forma, que es posible obtener un metamodelo `EMF` a partir de uno `MOF`, y viceversa.

La pieza central del metamodelo es la metaclase `Flow`, que representa un flujo de navegación entre páginas. Un flujo estará compuesto por una serie de estados, que en el diagrama `EMF` se modelan mediante la metaclase abstracta `State`. Un flujo tiene solamente un estado inicial (en el que comienza la ejecución del mismo), pero puede tener más de un estado final.

Los estados se pueden clasificar en estados a partir de los cuales pueden partir transiciones (`TransitionableState`), y estados sin transiciones salientes. En este caso, los estados finales (de tipo `EndState`) y los estados de decisión (`DecisionState`) son los únicos de los que no parte ninguna transición ordinaria.

Los estados con transiciones salientes pueden ser o bien estados de acción (`ActionState`), o bien estados de tipo vista (`ViewState`), o subflujos (`SubflowState`). Cada una de estas metaclases representa a uno de los tipos de estado disponible en `SWF`. La semántica de cada tipo de estado es la indicada en la sección 9.3, en la que se daba una visión general del *framework* para la definición de flujos. En la figura 9.5 se muestra un extracto del metamodelo en el que aparecen la jerarquía de metaclases que modelan los tipos de estado. Note que `State` y `TransitonableState` son clases abstractas.

Una transición (`Transition`), asociada a un evento, hace que un flujo cambie de un estado a otro. Cualquier estado de tipo traspasable, es decir, del que pueden partir

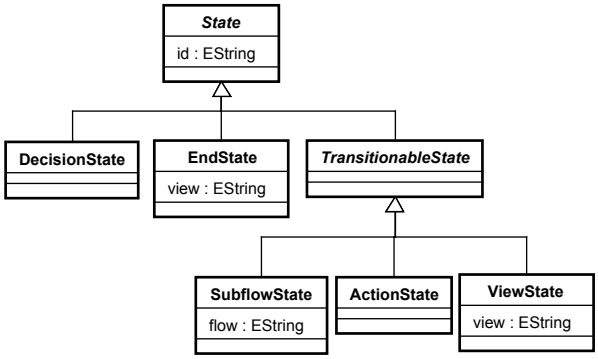


Figura 9.5: Extracto del metamodelo con la jerarquía de metaclases que modelan los estados de Spring Web Flow

transiciones, define un conjunto de una o más transiciones ordinarias que indican un camino hacia otro estado. La condición (metaclase `Condition`) asociada a un estado de decisión también provoca un cambio de estado. En este caso, pueden existir dos estados destino, el llegar a uno u otro dependerá de la evaluación de la propia condición. Este hecho se refleja en las referencias `tTest` y `fTest` que relacionan a la metaclase `Condition` y a `State`.

Una transición puede ser una transición global o de un estado. Una transición global (`GlobalTransition`) está relacionada con el flujo, y se puede activar en cualquier estado del mismo. Si una transición se puede producir en todos los estados traspasables del flujo, entonces se define como una transición global. Se puede decir que con este tipo de transición se proporciona un mecanismo para no tener que repetir por cada estado la misma información.

Las transiciones modeladas con `StateTransition` se disparan cuando se produce el evento asociado a la transición. Es el tipo de transición que se define en cualquier máquina de estados. La figura 9.6 muestra la jerarquía de clases que modelan transiciones.

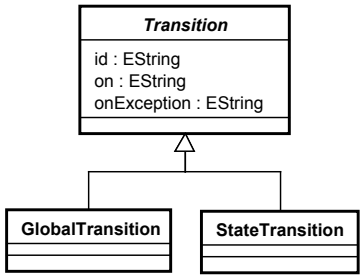


Figura 9.6: Extracto del metamodelo con la jerarquía de metaclases que modelan las transiciones de SWF

Una acción (`Action`) representa una funcionalidad a ejecutar en un punto determi-

nado del flujo. Los puntos en los que se pueden ejecutar acciones son: al iniciar el flujo; cada vez que se entra en un estado; justo antes de que se produzca el cambio de estado debido a que se dispara una transición; antes de dibujar un formulario o vista; cuando se sale de un estado, y cuando se termina el flujo. Esta semántica se ha modelado mediante las relaciones de composición entre `Action` y `Flow`, `Action` y `ViewState`, `Action` y `State`, y `Action` y `TransitionableState`, respectivamente.

Además de estos puntos, las acciones también se pueden asociar a los estados de tipo acción (`ActionState`), ya que es la forma de definir la lógica de negocio que se ejecuta en los mismos. Las acciones que se realizan antes de dibujar un formulario o vista, solamente se pueden ejecutar en estados de tipo `ViewState`, y serán acciones relacionadas con el *renderizado*, como por ejemplo, la configuración de los datos de un formulario.

Además de agrupar las acciones atendiendo a los puntos concretos donde se pueden lanzar, éstas se pueden clasificar en los siguientes tipos (figura 9.7): `SimpleAction`, `EvaluationAction`, `BeanAction` y `SetAction`. La acción simple se utiliza para encapsular acciones ligeras. Una acción de tipo `EvaluationAction` se utiliza cuando se quiere o bien invocar a un *bean* en un flujo, o bien evaluar alguna expresión de un flujo. Una acción de tipo *bean* se utiliza para invocar a un método de un objeto de los conocidos como tradicionales (lo que en la bibliografía se conoce como Plain Old Java Object (`POJO`)). Por último, las acciones de tipo `SetAction` se utilizan para asignar atributos en el flujo.

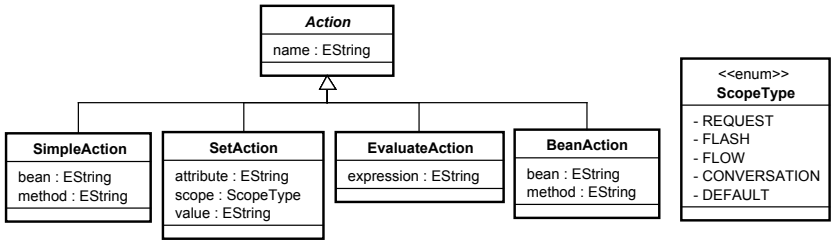


Figura 9.7: Extracto del metamodelo con la jerarquía de metaclases que modelan las acciones de SWF

Además de estas clases, existen otras que se pueden considerar como secundarias, porque no expresan los conceptos centrales que aporta el *framework*, y que se utilizan para definir casos muy específicos en los flujos. La metaclase `Attribute` se utiliza para definir atributos en un flujo, un estado o una transición. `ExceptionHandler` modela un manejador de excepciones asociado bien a un flujo, bien a un estado. `Variable` expresa una variable que se puede crear cuando se crea un flujo, `Argument` modela un argumento de un método definido en una acción e `Import` recoge aquellos recursos que pueden importarse en un flujo.

Finalmente, la metaclase `AttributeMapper` representa una correspondencia entre los atributos de un flujo y un subflujo. Se utiliza para modelar el hecho de que cuando se

lanza un subflujo desde un flujo, a éste se le pueden pasar parámetros de entrada. Igualmente, cuando el subflujo termina, puede devolver información mediante parámetros de salida. Mediante `AttributeMapper` se define la correspondencia entre los atributos del flujo y del subflujo. Como tal, solamente estará relacionada con `SubflowState`.

Además de las metaclases, para completar el metamodelo se han definido una enumeración (`ScopeType`) y restricciones `OCL`. La enumeración se utiliza para indicar los diferentes ámbitos que puede tener una variable del flujo. Mientras que las restricciones se han definido para evitar conseguir que dos atributos sean excluyentes, es decir, que si uno tiene un valor, el otro no lo puede tener. En el listado 9.1 se representan las restricciones `OCL` asociadas a los elementos del metamodelo. Como puede comprobarse, las restricciones están asociadas a las metaclases `Mapping`, `Variable` y `Transition`. La invariante asociada a `Mapping` expresa que los atributos `target` y `targetCollection` no pueden usarse a la vez. La invariante asociada a `Variable` representa una restricción similar, pero relacionada con los atributos `bean` y `class`, mientras que la asociada a `Transition` indica que las propiedades `on` y `onException` no pueden ser asignadas de forma simultánea. El metamodelo completo se puede ver en la figura 9.8 y una descripción detallada del metamodelo se puede encontrar en el informe interno [318].

Listado 9.1: Restricciones OCL asociadas al metamodelo de Spring Web Flow

```
1  context Mapping
2    inv: (self.target = null or self.target.size()=0) xor
3      (self.targetCollection = null or self.targetCollection.size()=0)
4
5  context Variable
6  inv: (self.class = null or self.class.size()=0) xor
7      (self.bean = null or self.bean.size()=0)
8
9  context Transition
10 inv: (self.on = null or self.on.size()=0) xor
11     (self.onException = null or self.onException.size()=0))
```

9.3.2 Una sintaxis gráfica concreta para SWF

Un metamodelo se puede considerar como la sintaxis abstracta de un lenguaje. La sintaxis concreta se puede definir de forma textual o de manera gráfica. En nuestro caso, se ha definido un lenguaje visual para representar los flujos de Spring Web Flow. El lenguaje visual está basado en la notación que define `UML` para los diagramas de estado, con las siguientes peculiaridades o adaptaciones: se han creado cinco estereotipos asociados a los estados definidos en los diagramas de estado `UML`. Cada uno de ellos representa uno de los estados tipo definidos en Spring Web Flow (figura 9.9). Cada estado se representa, por tanto, por un rectángulo con las esquinas redondeadas y su correspondiente estereotipo. La única excepción son los estados finales que se representan con dos círculos concéntricos, de los cuales, el más interno aparece relleno. Algunos de estos estados (`ViewState`, `ActionState` y `SubflowState`), se representan con el rectángulo dividido

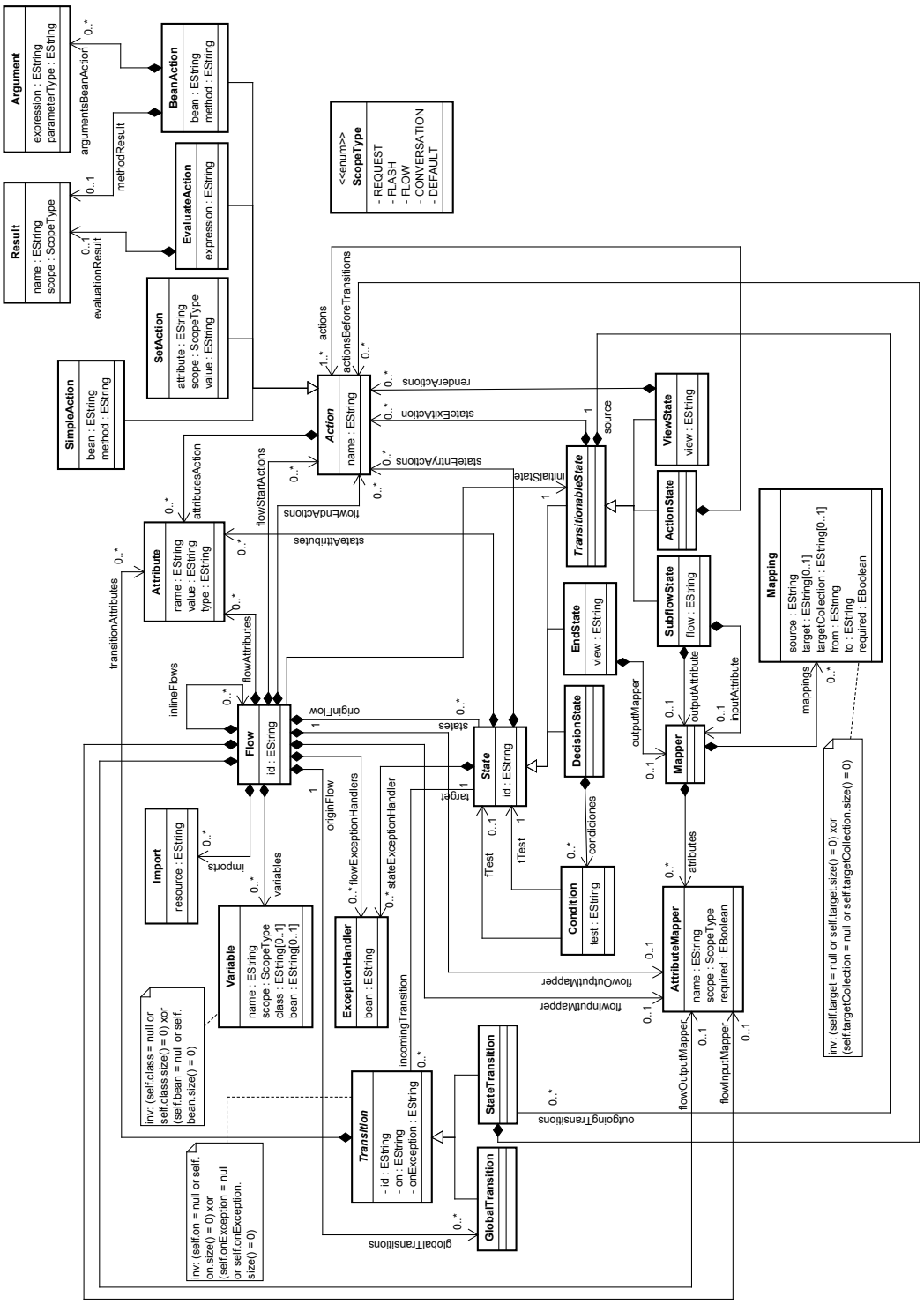


Figura 9.8: Un metamodelo para Spring Web Flow

en dos partes. En la parte superior se muestra el nombre del estado y el estereotipo, que representa el tipo de estado, mientras que en la parte inferior se indica, mediante un estereotipo, la propiedad más representativa del mismo. Así, en los estados tipo vista se representa la vista (`<<view>>`), en los de acción, la acción(`<<action>>`), y finalmente, en los subflujos (`<<flow>>`), el subflujo al que se accede. El estado final también puede tener asociada una vista, que es la pantalla o interfaz que se muestra al terminar el flujo.

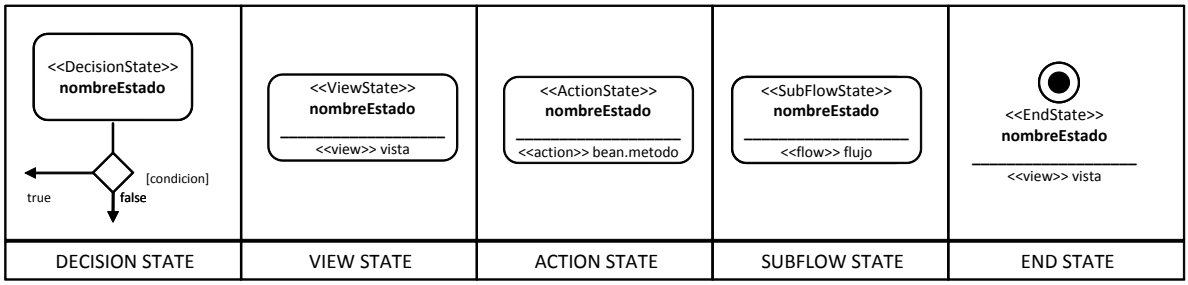


Figura 9.9: Notación visual para los tipos de estados definidos en SWF

En la figura 9.10 se representan los distintos tipos de transiciones que nos podemos encontrar. Así, una transición normal se representa con una flecha que parte del estado origen y se dirige hacia el estado destino, acompañada por una etiqueta que da nombre a la transición. Una transición global se representa por una flecha que parte de la marca de inicio del flujo (un círculo pequeño relleno) hacia el estado destino, acompañada también de una etiqueta que indica el nombre de la transición. Una transición de tipo excepción se representa igual que una transición normal estereotipada con `<<exception>>`. Una transición dinámica se representa igual que una transición normal, con la diferencia de que en este caso, el rectángulo que representa el estado destino contiene un signo de interrogación (?) y una expresión que sirve para calcular cuál será el estado destino. Finalmente, una transición condicional se representa por un rombo hueco del que parten dos bifurcaciones, una con la etiqueta `true`, y otra con la etiqueta `false`. El rombo va acompañado de una etiqueta que contiene la condición a evaluar y que determinará el estado final. La rama `false` es opcional, es decir, puede no aparecer. Este tipo de transición se puede ver en el estado de decisión dibujado en la figura 9.9.

Una acción o conjunto de acciones se representa por un rectángulo acompañado de una línea discontinua por la que queda unida a otro elemento del flujo (figura 9.11). Así, se puede asociar a un estado (figura 9.11, tercer recuadro), a una transición (figura 9.11, segundo recuadro), o a la marca de inicio del flujo (figura 9.11, cuarto recuadro). En este último caso, se representa a una acción a nivel del flujo. Dentro de la lista de acciones pueden aparecer varios estereotipos. En las acciones asociadas a los estados de tipo vista (figura 9.11, primer recuadro) puede aparecer `<<render>>`, y en las acciones asociadas al flujo y a los estados pueden aparecer bien `<<start>>`, bien `<<end>>`, diferenciando

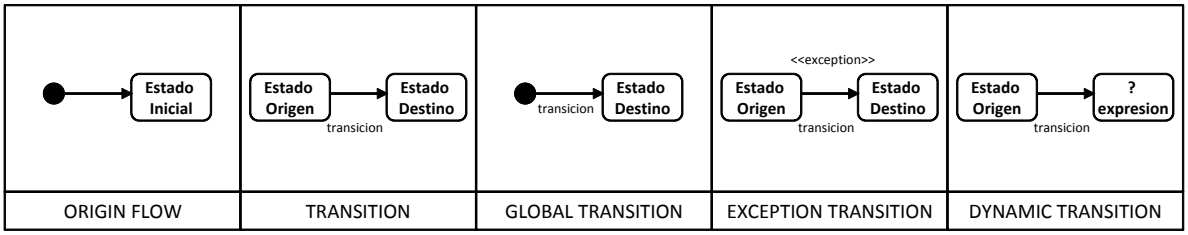


Figura 9.10: Notación visual para los tipos de transiciones definidos en SWF

así entre las acciones que se han de ejecutar al entrar o al salir del flujo o el estado, respectivamente.

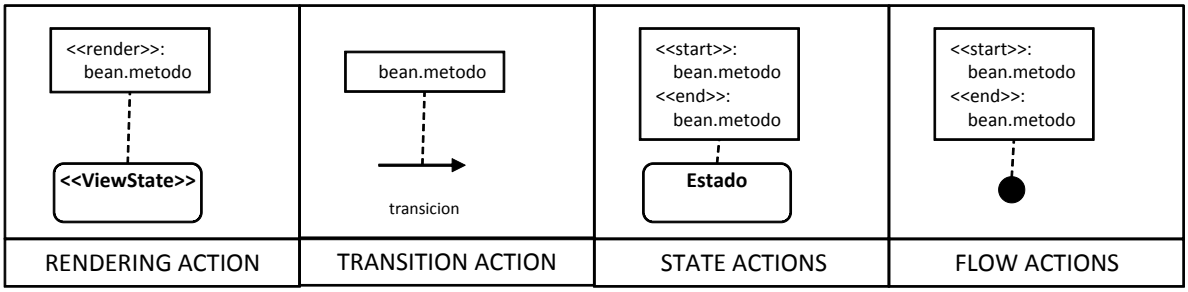


Figura 9.11: Notación visual para las acciones de SWF, dependiendo del punto del flujo donde sean definidas

9.3.3 Transformaciones de modelo a texto

Las transformaciones modelo a texto se han implementado con MOFScript [312], porque en el momento de desarrollar esta tesis aún no hay disponible una herramienta potente que sea conforme al estándar MOF Model to Text Transformation de la `OMG`. MOFScript es parte del proyecto Eclipse llamado Generative Modeling Technologies (`GMT`) [98], un proyecto cuyo principal objetivo era producir una serie de prototipos en el área de la Ingeniería Dirigida por Modelos, y que ha servido como incubadora para el proyecto de modelado de Eclipse (Eclipse Modeling Project (`EMP`) [95]). MOFScript comprende tanto un *plugin* de Eclipse como un lenguaje para especificar transformaciones de modelo a texto.

Los ficheros MOFScript tienen la extensión `.m2t`. Se compilan, y luego se pueden ejecutar. El resultado de una trasformación está formado normalmente por un conjunto de ficheros. En nuestras transformaciones de modelo a texto para Spring Web Flow, el resultado se recoge en un archivo `XML` que especifica la definición del flujo. Si el lector está interesado en obtener el archivo fuente con la definición de estas trans-

formaciones, puede acceder a la URL `http://www.lsi.us.es/~reinaqu/org.mwacsl/springWF/transformations/SWF2XML.m2t` para descargárselo.

La transformación se llama `SpringWF2XML` y tiene un parámetro de entrada que hace referencia al modelo de entrada. Este modelo tiene que ser conforme al metamodelo de Spring Web Flow definido en la sección 9.3.1, y se supone que está libre de errores. Es decir, si el modelo se ha generado mediante un editor, todo el proceso de comprobación del modelo ha debido hacerse en este editor. Si el modelo se ha generado automáticamente a partir de un modelo a nivel de `PIM`, entonces la transformación modelo a modelo ha debido producir un modelo correcto. Los modelos Spring Web Flow conformes a nuestro metamodelo tienen la extensión `.springwf`. Si el lector está interesado, puede descargarse una serie de modelos para comprobar la transformación de la siguiente dirección: `http://www.lsi.us.es/~reinaqu/org.mwacsl/springWF/models/`. Uno de los ejemplos que aparece en esta página es el representado en la figura 11.2, el resto de flujos se han realizado a partir de los ejemplos que acompañan a Spring Web Flow.

Una transformación MOFScript está formada por un conjunto de reglas. Una de ellas debe ser marcada como punto de entrada. El punto de entrada define el lugar en el que la transformación comenzará su ejecución. La idea es similar al método `main` de las aplicaciones Java. El punto de entrada puede tener un contexto. La idea del contexto es similar a la idea de contexto `OCL`. El contexto determina qué elemento del metamodelo será el punto inicial de la transformación. En nuestro caso, el punto inicial es la metaclase `Flow`. El listado 9.2 muestra el punto de entrada y la cabecera de esta transformación.

Listado 9.2: Cabecera de la transformación y punto de entrada

```
1  texttransformation SpringWF2XML (in flow: "http://www.mwacsl.com/
       springWF/1.0") {
2
3   flow.Flow::main(){
4      file (self.id+ ".xml")
5      <%<?xml version="1.0" encoding="UTF-8"?>%>
6      <%<flow xmlns="http://www.springframework.org/schema/webflow"%>
7      <%"http://www.springframework.org/www.w3.org/2001/XMLSchema-
           instance">%>
8      <%xsi:schemaLocation="%>
9      <%http://www.springframework.org/schema/webflow%>
10     <%http://www.springframework.org/schema/webflow/spring-webflow-1.0.
           xsd">%>
11     self.flowDefinition(1)
12     <%</flow>%>
13    } // end of main
```

Las reglas son similares a las funciones. Pueden tener un contexto (el elemento del metamodelo para el que se define la regla), un valor de retorno, y un cuerpo que contendrá un conjunto de sentencias. Tanto el contexto como el valor de retorno son opcionales. La tabla 9.2 muestra un resumen con la signatura de las reglas que se

han definido en nuestra transformación `SpringWF2XML`. La tabla está dividida en dos bloques. En cada bloque, la primera columna representa un elemento del metamodelo (el contexto), mientras que la segunda representa el conjunto de reglas asociadas con ese elemento.

Metaclase	Regla	Metaclase	Regla

Flow	`main()`
	`flowDefinition(prof:integer)`
	`sectionFlowVariables(prof:integer)`
	`sectionFlowAttributes(prof:integer)`
	`sectionFlowInputMapper(prof:integer)`
	`sectionFlowOutputMapper(prof:integer)`
	`sectionFlowGlobalTransitions(prof:integer)`
	`sectionExceptionHandlerFlow(prof:integer)`
	`sectionImport(prof:integer)`
	`sectionInlineFlow(prof:integer)`
	`sectionFlowStartActions(prof:integer)`
	`sectionFlowEndActions(prof:integer)`
	`sectionFlowStates(prof:integer)`
		DecisionState	`generateDecisionState(prof:integer)`

		Transition	
		StateTransition	`generateTransitionActions(prof:integer)`
		GlobalTransition	`generateTransition(prof:integer)`
		Attribute	`generateAttribute(prof:integer)`
		Variable	`generateVariable(prof:integer)`
		Action	`generateActionAttributes(prof:integer)`
		SetAction	`generateAction(prof:integer)` `generateSetAction(prof:integer)`
		SimpleAction	`generateSimpleAction(prof:integer)`
		EvaluateAction	`generateEvaluateAction(prof:integer)`
			`generateResultEvaluateAction(prof:integer)`

State	`sectionStateAttributes(prof:integer)`
	`sectionStateEntryActions(prof:integer)`
	`generateState(prof:integer)`
	`sectionStateExceptionHandlers(prof:integer)`
		BeanAction	`generateBeanAction(prof:integer)`
			`generateMethodArguments(prof:integer)`
			`generateMethodResult(prof:integer)`

		Argument	`generateArgument(prof:integer)`
TransitionableState	`sectionStateTransitions(prof:integer)`	Result	

	`sectionStateExitActions(prof:integer)`
	`generateTransitionableState(prof:integer)`
		Mapper	`generateInputAttributes(prof:integer)`
			`generateOutputAttributes(prof:integer)`
			`generateMappings(prof:integer)`

EndState	`generateEndState(prof:integer)`		
ActionState	`generateActionState(prof:integer)`	Mapping	`generateMapping(prof:integer)`
ViewState	`generateViewState(prof:integer)`	AttributeMapper	

SubflowState	`generateSubflowState(prof:integer)`
	`sectionSubflowInputMapper(prof:integer)`
	`sectionSubflowOutputMapper(prof:integer)`
		Condition	`generateCondition(prof:integer)`
			`generateTrueCondition(prof:integer)`

		Import	`generateImport(prof:integer)`

Tabla 9.2: Resumen de reglas de la transformación modelo a texto

Para simplificar la forma de proporcionar la salida a un dispositivo, MOFScript permite el uso de caracteres de escape, de forma similar a como se hace en la mayoría de los lenguajes de *script*. En las reglas, el comienzo de la salida con caracteres de escape se indica con la etiqueta <% y el final con %>. En el listado 9.2 se muestran algunos ejemplos del uso de estos caracteres de escape. Así, por ejemplo, al ejecutarse la línea 5 del listado, en el dispositivo de salida se escribirá exactamente lo que está entre las etiquetas <% y %>, es decir: `<?xml version="1.0" encoding="UTF-8"?>`

Si nos fijamos en la tabla 9.2, se puede ver que hay algunos elementos del metamodelo que no tienen asociada ninguna regla. Esto ocurre porque lo que hay que generar a partir de ellos depende de la referencia que ha conducido hasta los mismos. En otras palabras, lo que se genera depende de la ruta de navegación que se ha seguido en el metamodelo hasta llegar al elemento. Por ejemplo, si echamos un vistazo a la metaclase `Result` (ver figura 9.8), se puede comprobar que hay dos relaciones de composición diferentes que nos pueden guiar hasta ella: podemos tener el resultado de evaluar una acción (relación entre `Result` y `EvaluateAction`), o bien, el resultado de evaluar un método (relación entre `Result` y `BeanAction`). Las diferencias se han tratado en las reglas asociadas a los contenedores `EvaluateAction` y `BeanAction`. Así, en el listado 9.3 se muestra cómo desde las reglas asociadas a `EvaluateAction` y `BeanAction` hay una invocación a las reglas `generateResultEvaluateAction` (línea 10) y `generateMethodResult` (línea 29), respectivamente. Dichas reglas se encargan de generar el código asociado a `Result`,

que no es el mismo si se genera desde un punto o desde otro.

Listado 9.3: Reglas generateEvaluateAction y generateBeanAction

```
1  flow.EvaluateAction::generateEvaluateAction(prof:integer){
2      tabulador (prof)
3      <%<evaluate-action expression="%> self.expression
4      if (self.evaluationResult == null &&
5          self.attributesAction.isEmpty())
6          <%"/> %>
7      else {
8          <%"> %>
9          self.generateActionAttributes(prof+1)
10         self.generateResultEvaluateAction(prof+1)
11         <%</evaluation-action>%>
12   }
13 }//end generateEvaluateAction
14
15 flow.BeanAction::generateBeanAction(prof:integer){
16     tabulador(prof)
17     <%<bean-action bean="%> self.bean <%" %>
18     if (self.method != null){
19       <%method="%>self.method
20       if (self.attributesAction.isEmpty() &&
21           self.argumentsBeanAction.isEmpty() &&
22            self.methodResult == null){
23         <%"/> %>
24       }
25      else {
26         <%"> %>
27         self.generateActionAttributes(prof+1)
28         self.generateMethodArguments(prof+1)
29         self.generateMethodResult(prof+1)
30         tabulador(prof)
31         <%</bean-action>%>
32       }
33     }
34   }//end generateBeanAction
```

Las reglas también pueden tener cualquier número de parámetros. En nuestra transformación todas las reglas definidas tienen un parámetro llamado `prof`, tal y como se puede comprobar en la tabla 9.2. Este parámetro se usa para darle formato al texto de salida e indica el nivel de profundidad del elemento. El nivel de profundidad determinará el número de tabulaciones a imprimir antes de la línea de salida. Como hay muchas reglas, en este apartado solamente se comentarán aquellas relacionadas con los principales elementos del metamodelo: `Flow`, `State` y `Transition`.

La regla `flowDefinition` (listado 9.4) genera todos los componentes del flujo. Por cada relación de composición de la metaclase `Flow` con otra metaclase, se ha definido una regla. `flowDefinition` se encarga de imprimir el estado inicial y de llamar a

las reglas asociadas con los componentes del flujo. Una de estas reglas intermedias es `sectionFlowStates`.

Listado 9.4: Regla flowDefinition

```
1  flow.Flow::flowDefinition(prof:integer){
2      tabulador(prof)
3      <%<start-state idref="%> self.initialState.id<%"/>%>
4
5      self.sectionFlowAttributes(prof)
6      self.sectionFlowVariables(prof)
7      self.sectionFlowInputMapper(prof)
8      self.sectionFlowStartActions(prof)
9      self.sectionFlowStates(prof)
10     self.sectionFlowEndActions(prof)
11     self.sectionFlowOutputMapper(prof)
12     self.sectionFlowGlobalTransitions(prof)
13     self.sectionImport(prof)
14     self.sectionInlineFlow(prof)
15 }//end flowDefinition
```

`sectionFlowStates` (listado 9.4, línea 9) comprueba el tipo de estado y, dependiendo del estado, llama a una regla concreta para el tipo de estado. Por ejemplo, la regla `generateEndState` imprime las propiedades de un estado final, y `generateViewState` trata con un estado de tipo vista.

Un estado de tipo vista (`ViewState`) tiene ciertas particularidades con respecto a otros estados (por ejemplo, tiene una propiedad que representa la vista a mostrar y es única para este tipo de estados); pero también tiene ciertas propiedades en común con los otros tipos de estado. Para imprimir estas propiedades comunes se invoca a la regla `generateState` justo después de mostrar las particularidades (listado 9.5, línea 8). Luego se generan las acciones relacionadas con el *renderizado* de la vista (líneas 10-18), para finalmente, invocar a la regla `generateTransitionableState` (línea 19).

Listado 9.5: Regla generateViewState

```
1  flow.ViewState::generateViewState(prof:integer){
2      tabulador(prof)
3      <%<view-state id="%>self.id <%"%>
4      if (self.view != null){
5          <% view="%>self.view<%"%>
6      }//endif
7      <%>%>
8      self.generateState(prof+1)
9      //generate acciones de renderizado
10     if (!self.renderActions.isEmpty()){
11       tabulador(prof+1)
12       <%<render-actions>%>
13       self.renderActions->forEach(act:flow.Action){
14       act.generateAction(prof+1)
```

```
15        }//foreach
16        tabulador(prof+1)
17        <%</render-actions>%>
18      }//endif
19      self.generateTransitionableState(prof+1)
20      tabulador(prof)
21      <%</view-state>%>
22  }//generateViewState
```

La regla `generateTransitionableState` (listado 9.6) produce la serialización de las transiciones entre estados (mediante la invocación de la regla `sectionStateTransitions`), y, a continuación, las acciones de salida (usando la regla `sectionStateExitActions`)

Listado 9.6: Regla generateTransitionableState

```
1 flow.TransitionableState::generateTransitionableState(prof:integer){
2     self.sectionStateTransitions(prof)
3     self.sectionStateExitActions(prof)
4 }//generateTransitionableState
```

9.4 Metamodelo de JSP

Teniendo en mente la reutilización como uno de los principios de desarrollo de software, el metamodelo de Java Server Pages (`JSP`) utilizado para generar código para esta plataforma es el definido dentro del proyecto Modisco [167]. El metamodelo se ha definido como una extensión del metamodelo de `XML` [168], que modela un subconjunto de los conceptos definidos en el estándar W3C XML Recommendation [401]. En la figura 9.12 se muestra el metamodelo de `JSP`. Las clases que pertenecen al metamodelo de `XML` aparecen estereotipadas con `<<from xml>>`.

La raíz del metamodelo es la metaclase `Model`. Un modelo `JSP` representa un conjunto de páginas, por lo que se puede modelar desde una página `JSP` hasta un conjunto de páginas de un sitio web. Cada página `JSP` se modela mediante la metaclase `Page`. Una página está compuesta por un conjunto de nodos (`Node`), de la misma forma que cualquier archivo `XML`, ya que `JSP` usa el mismo mecanismo de etiquetas, atributos y comentarios.

La mayoría de las metaclases específicas de `JSP` se han especificado como una extensión de la metaclase `Element`, y como se indica en la especificación "Java Server Pages Specifications Version 1.2" [254], el contenido de un archivo `JSP` se divide en cuatro categorías, que se han modelado mediante las metaclases `JSPScript`, `JSPAction`, `JSPDirective` y `JSPComment`. La figura 9.12 se ha enriquecido con anotaciones asociadas a las metaclases propias de `JSP` en las que se muestran ejemplos del tipo de elemento que representa. Así, por ejemplo, la metaclase `JSPComment` tiene asociada una anotación `<%-- Comentario JSP --%>`, mostrando un ejemplo de este tipo de comentario.

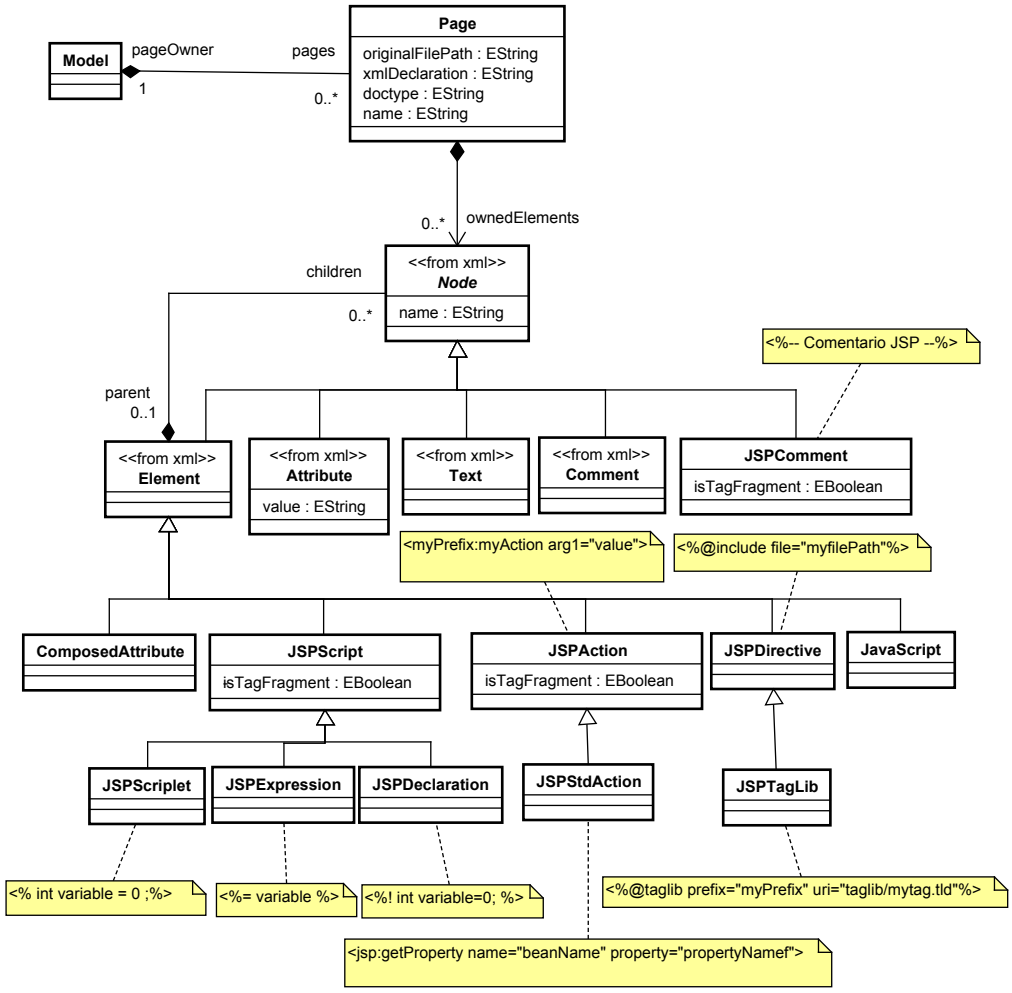


Figura 9.12: Metamodelo de JSP desarrollado dentro del proyecto Modisco

9.5 Sumario

En este capítulo se ha presentado tres metamodelos para tratar con la navegación en tres plataformas específica: `HTML`, Spring Web Flow y `JSP`. Además, la plataforma Spring Web Flow se ha definido una sintaxis gráfica concreta. Una primera versión de este metamodelo, así como la sintaxis concreta se presentaron en [334]. Además, se han definido una serie de transformaciones de modelo a texto que permiten obtener archivos `XML` con la definición de flujos listos para ser utilizados en la plataforma Spring Web Flow. Estas transformaciones, así como una revisión del metamodelo se publicaron en [321]. También se ha presentado una transformación modelo a modelo que genera un modelo de `HTML` que contiene un esqueleto de la navegación estructural y de utilidad. Esta transformación se ha publicado en [329].

Un problema importante que surge cuando se trabaja con *frameworks* es que el tiempo de lanzamiento de nuevas versiones de los mismos es cada vez más corto. Este problema se ve acentuado en aquellas plataformas de software libre, debido a la gran participación de los usuarios en el proceso, testeando los programas, descubriendo errores e informando a los desarrolladores de los mismos. En este sentido, el trabajo con estas plataformas, abre nuevas líneas de aplicación del `DSDM`. Trabajos iniciales de esta línea se han presentado en [334] y en [335].

The question of the composition of perceptible objects is one which already occupied the mind of the ancient Greeks.

Johannes Stark

10 Combinación de Concerns

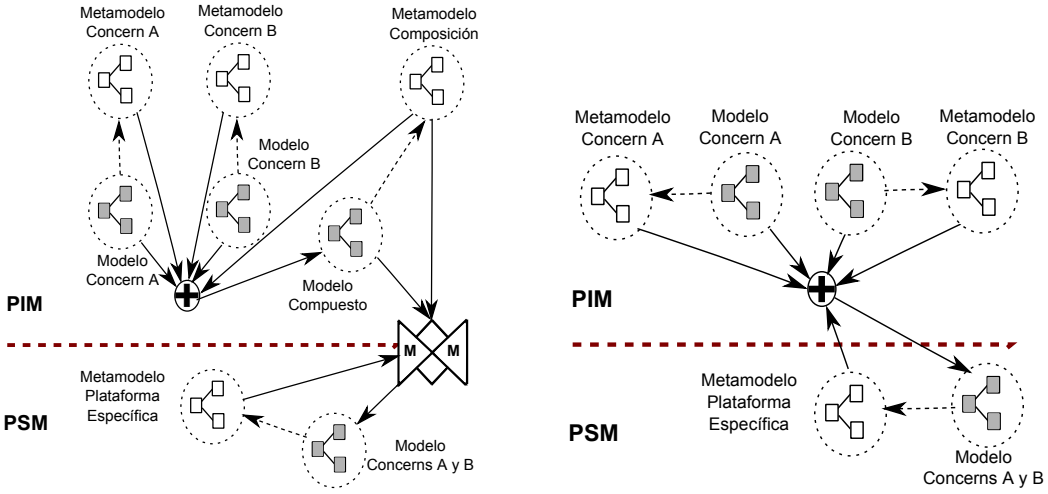
Este capítulo se centra en estudiar distintas estrategias para combinar los concerns *que en los capítulos anteriores se han definido de forma separada mediante CSL's a nivel PIM. Para ello el capítulo se ha estructurado de la siguiente forma: la sección 10.1 introduce cómo se va a abordar el problema y apunta la tecnología que se va a usar para resolverlo. La sección 10.2 da una visión de ATLAS Model Weaver (AMW), la tecnología escogida para realizar la composición de modelos como base para entender el resto de las secciones. La sección 10.2 explica cómo se realiza el* weaving *de modelos con ATLAS Model Weaver. La sección 10.3 explica las distintas estrategias de composición de* concerns *detectadas en* `MWACSL`*, mientras que la sección 10.4 se dan detalles concretos de cómo se han implementando estas estrategias para los distintos* concerns *navegacionales separados en el capítulo 8. Finalmente, se presenta un sumario del capítulo.*

10.1 Introducción

En `MWACSL` los *concerns* se modelan mediante lenguajes específicos de dominio, así que el proceso de composición de *concerns* se puede ver como un proceso de composición de modelos. Según [393], el proceso de composición de modelos es el proceso de integrar una serie de modelos, bien asegurando su interoperabilidad como modelos autónomos, bien combinándolos en un nuevo modelo compuesto.

El asumir esta definición implica que la composición de modelos no siempre produce un modelo compuesto, sino que se pueden dejar los modelos originales autónomos y

asegurar su interoperabilidad. Esta cuestión es especialmente importante en el caso de sistemas complejos en los que combinar todos los modelos en uno puede dar lugar a un resultado poco manejable y entendible. Si trasladamos estas dos definiciones al marco de MWACSL, nos pueden quedar las dos opciones que se muestran en la figura 10.1. La notación usada en esta figura está inspirada en la que usa Gronback en su libro [156], y es la que se usará en el resto del capítulo. Un listado detallado de los iconos usados en esta notación y su significado se presenta en la figura 10.2.



(a) Composición a nivel de PIM

(b) Composición a nivel de PSM

Figura 10.1: Procesos de composición

La figura 10.1(a), representa lo que hemos llamado composición a nivel de PIM. En este caso, el modelo compuesto se hace explícito y se sitúa a nivel de PIM, lo que implica que hay que definir un metamodelo para el lenguaje en el que los *concerns* aparecen compuestos. El modelo a nivel de PSM se obtiene mediante una transformación de modelo a modelo. Note que en el modelo a nivel de PSM están mezclados los *concerns*.

La figura 10.1(b), es lo que hemos denominado composición a nivel de PSM. En este caso no hay un modelo explícito de composición, sino que el modelo compuesto se obtiene directamente a nivel de PSM, por lo que el metamodelo para este lenguaje compuesto es el propio metamodelo de la plataforma, ya que en este lenguaje, los *concerns* ya están mezclados. En este caso, es necesario que la herramienta usada para realizar la composición permita realizar una fase de traducción de los lenguajes de *concerns* independientes de la plataforma al lenguaje específico de la plataforma, ya que no solamente se están componiendo los *concerns*, sino que el resultado se expresa en un lenguaje distinto, a un nivel de abstracción más bajo.

En MWACSL se ha elegido la segunda opción, es decir, escoger como metamodelo de lenguaje compuesto el propio metamodelo del lenguaje específico de plataforma. A la hora de escoger la herramienta para enfrentarse al proceso de composición ha sido determinante el hecho de trabajar con varios metamodelos, ya que cuando los modelos a

componer son conformes a diferentes metamodelos, el proceso de composición de modelos ha de incluir una fase de traducción [200], que se encargue de traducir los modelos de entrada a los de salida. Este requisito solamente lo cumplen dos herramientas [199]: Atlas Model Weaver (AMW) [85] y Epsilon Merging Language (EML) [106]. De estas dos opciones, se ha optado por AMW, con lo que el esquema de composición de modelos representado por el operador $\bigoplus$ en la figura 10.1(b), se traduce por la creación de un modelo de *weaving*, que sirve como especificación de la composición, más una transformación modelo a modelo que es a través de la que se obtiene el modelo compuesto.

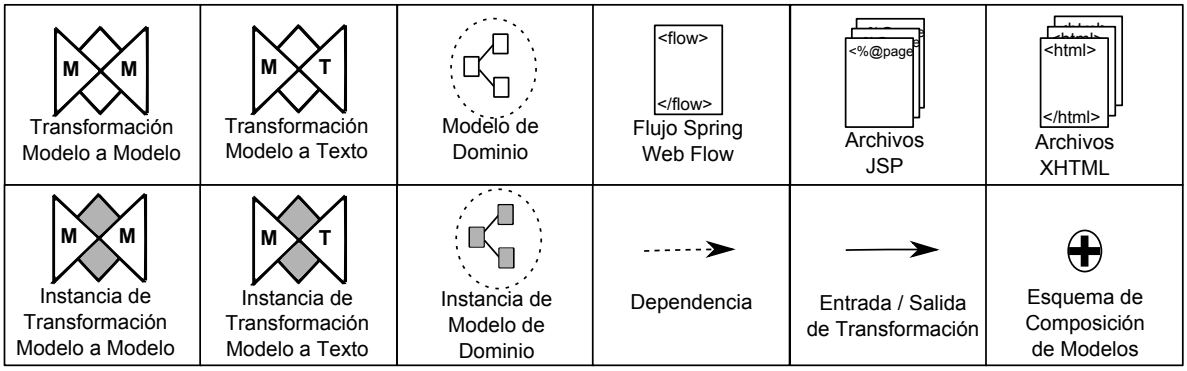


Figura 10.2: Notación usada en el resto de secciones [156]

10.2 Weaving de modelos en AMW

Los modelos de *weaving*, según [85, 84], se han concebido para definir relaciones (o correspondencias) entre elementos de modelos distintos. Estas correspondencias se pueden utilizar en un gran dominio de problemas distintos. En cada dominio, las relaciones entre los elementos de los modelos tendrán una semántica concreta. La forma de enfrentarse a esta variabilidad en AMW es definir un metamodelo común que sirva como núcleo, llamado *core weaving metamodel*, y extenderlo para cada uno de los dominios concretos. En la figura 10.3 se muestra este metamodelo común. Como podrá observar al aparecer los nombres en cursiva, todas las metaclases son abstractas, por lo que, forzosamente habrá que extenderlo para poder utilizarlo en un dominio concreto.

`WElement` es una metaclase base de la que heredan el resto de las metaclases. Tiene dos propiedades: un nombre (`name`) y una descripción (`description`). La raíz de un modelo de *weaving* es la metaclase `WModel`, que está compuesta por elementos del propio modelo de weaving (`ownedElements`) y referencias a los modelos a componer (`wovenModel`). El número de modelos a componer dependerá del dominio concreto en el que se usa. `WLink` representa una relación o correspondencia entre elementos del modelo y

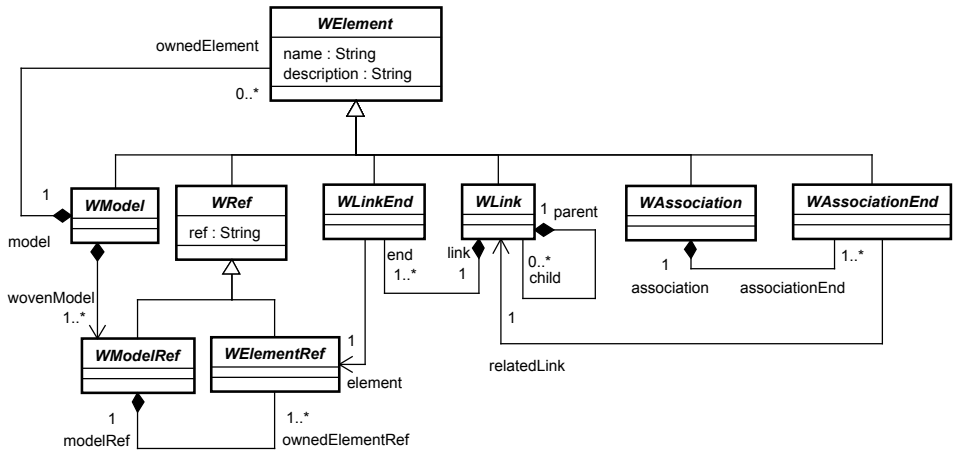


Figura 10.3: Núcleo de metamodelo de weaving propuesto en AMW y extraído de [83]

`WLinkEnd` representa a los extremos de un enlace. Las asociaciones entre relaciones se modelan mediante la metaclase `WAssociation`, mientras que `WAssociationEnd` representa un extremo de la asociación, de forma similar a `WLink`.

`WElementRef` representa un identificador único o referencia que se asocia a cada uno de los elementos que se van a relacionar. Este identificador se guarda en el atributo `ref` de la metaclase `WRef`, de la que heredan tanto `WElementRef` como `WModelRef`, que representa una referencia a un modelo completo.

Una de las extensiones que se han definido al núcleo del metamodelo de weaving se muestra en la figura 10.4. Esta extensión [83] se usa para definir anotaciones a modelos para incluir información que no está definida en el metamodelo y que no es conceptualmente relevante para el mismo. En la figura 10.4, las metaclases que pertenecen al núcleo del metamodelo de *weaving* aparecen con el estereotipo `<<from mwcore>>`, mientras que las metaclases que forman parte de la extensión no están estereotipadas y se han resaltado con un fondo más oscuro.

En este caso, la raíz del metamodelo de anotación pasa a ser la metaclase `AnnotationModel`, que restringe el número de modelos que se van a relacionar, ya que mientras que la relación de composición entre `WModel` y `WModelRef` tiene cardinalidad `1..*` (un modelo de *weaving* puede relacionar uno o varios modelos), en el caso concreto de un modelo de anotación, `AnnotationModel` esta compuesto por exclusivamente un modelo anotado (`AnnotatedModel`). La misma restricción se aplica a las anotaciones, ya que una anotación solamente se puede aplicar a un elemento del metamodelo que se va anotar.

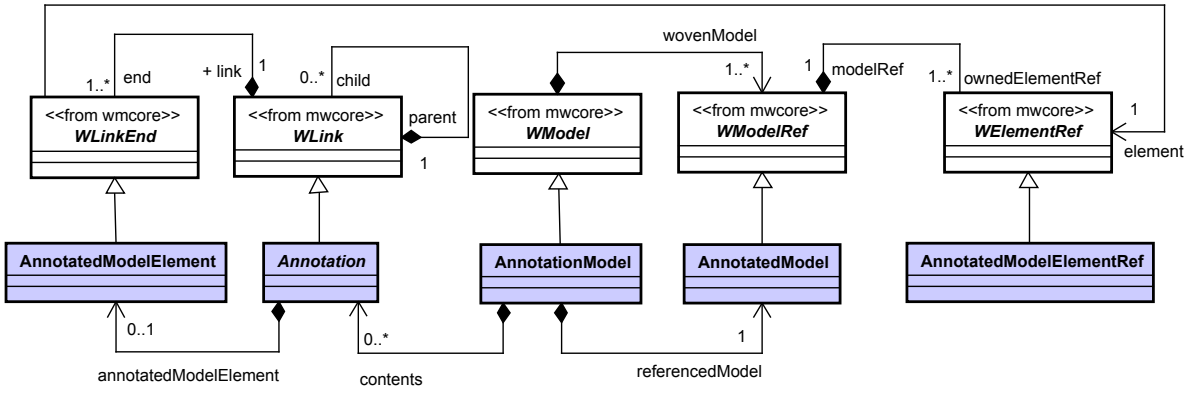


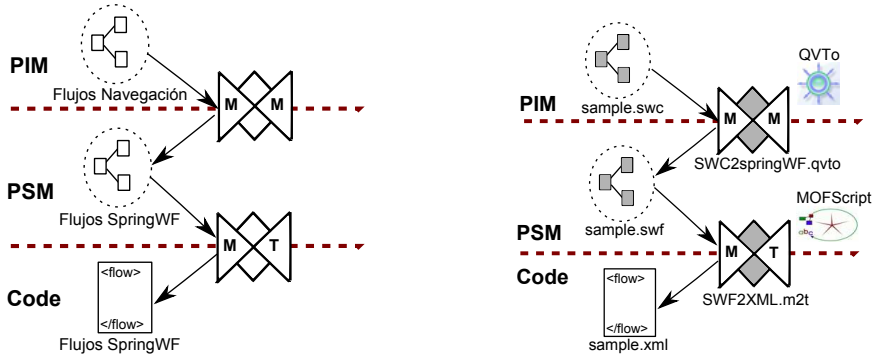
Figura 10.4: Metamodelo de anotación

10.3 La composición de *concerns* en MWACSL

Dependiendo de los *concerns* y de la propia plataforma de implementación, se han detectado diferentes estrategias para obtener modelos a nivel de PSM a partir los *concerns* modelados a nivel de PIM. Estas estrategias van desde la transformación directa, en el caso en que el *concern* también aparezca separado en la plataforma, a la composición, bien asimétrica, bien simétrica, en el caso en que en la plataforma concreta aparezcan los *concerns* mezclados. En los siguientes subapartados se detallan estas estrategias.

10.3.1 Transformación directa

Este tipo de transformación tiene lugar cuando un concepto se mantiene separado en todos los niveles de modelado (PIM, PSM y código). Un esquema de este tipo de transformación es el que se muestra en la figura 10.5, que se utiliza cuando se quiere generar un flujo de Spring Web flow a partir de un modelo de flujo de navegación (SWC). Spring Web Flow es un *framework* que se centra exclusivamente en los flujos de navegación controlada, por lo que la definición de estos flujos se mantiene separada a todos los niveles. En este caso, a partir de un modelo SWC (definido a nivel de PIM) se genera un modelo de Spring Web Flow mediante una transformación modelo a modelo. Posteriormente, este modelo de Spring Web Flow, se transformará mediante una transformación modelo a texto a un archivo XML con la especificación del flujo de transformación (figura 10.5(a)). Una instancia de este esquema es la que se muestra en la figura 10.5(b), en la que la transformación modelo a modelo se ha implementado en QVT Operational Mapping y la transformación modelo a texto con MOFScript.



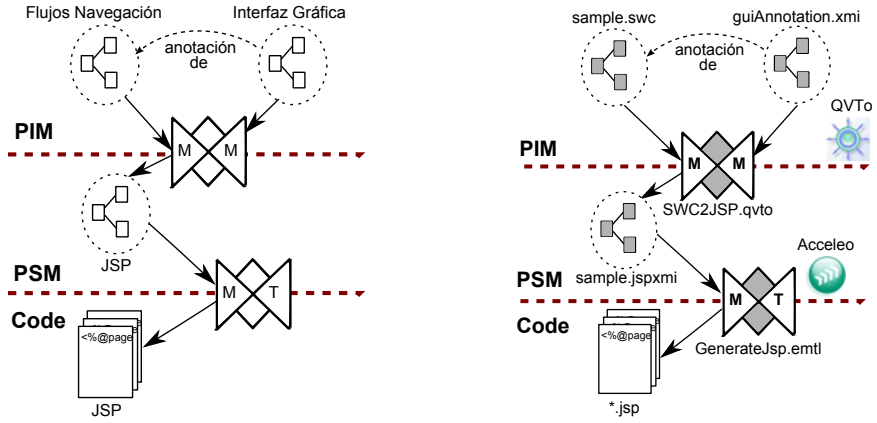
(a) Esquema

(b) Ejemplo de instancia del esquema

Figura 10.5: Esquema de transformación directa

10.3.2 Composición asimétrica

Este tipo de composición se aplica cuando uno de los modelos se define en base a otro, proporcionando información adicional sobre el mismo. La información que aporta el nuevo modelo no se incluye en el modelo original para mantener una mejor separación de competencias. En [222], Kolovos et al. denominan a este tipo de relación *aspectOf*. Un esquema de este tipo de composición se muestra en la figura 10.6, en la que se puede apreciar cómo están relacionados la interfaz gráfica y los flujos de navegación controlados. Note que en esta relación no intervienen la navegación estructural y de utilidad, ya que en los flujos de navegación controlada se quitan todos los elementos de navegación obligando al usuario a seguir el camino definido por el flujo o a cancelar la operación.



(a) Esquema

(b) Ejemplo de instancia del esquema

Figura 10.6: Esquema de composición asimétrica

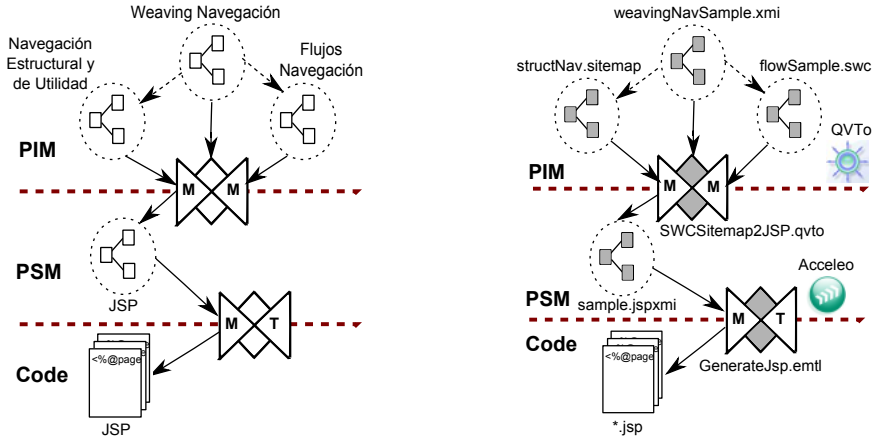
La figura 10.6(a) muestra un esquema de este tipo de composición, en el que las cuestiones relacionadas con la interfaz se han definido como un modelo de anotación sobre el modelo de navegación controlada. Ambos modelos (y sus correspondientes metamodelos) sirven como entrada a una transformación de modelo a modelo que genera un modelo para `JSP`. Note que el metamodelo para `JSP` es el metamodelo resultado de la composición. Finalmente, mediante transformaciones de modelo a texto se obtienen archivos `JSP` en los que se mezcla el flujo de navegación controlada y la interfaz de usuario. La figura 10.6(b) muestra una instancia de este esquema en el que la transformación modelo a modelo se ha implementado en QVT Operational Mapping y la transformación de modelo a texto está escrita en Acceleo, puesto que se han reutilizado los artefactos ya definidos para el proyecto Modisco [125].

10.3.3 Composición simétrica

Esta estrategia se aplica cuando se quiere enlazar dos modelos que inicialmente no están relacionados. Es el tipo de relación que en [222] Kolovos et al. clasifican como *weaves*. Un ejemplo de este tipo de composición se muestra en el esquema de la figura 10.7, en el que la relación entre los modelos de navegación estructural y de utilidad con los modelos de flujos se realiza mediante un modelo de *weaving*. Mediante una transformación de modelo a modelo se obtiene un modelo específico de plataforma en el que estos *concerns* aparecen mezclados. Note que, de nuevo, el metamodelo del lenguaje compuesto es el metamodelo de `JSP` usado a nivel de `PSM`. Estos modelos que representan archivos `JSP` servirán como entrada a una transformación modelo a texto que producirá los correspondientes artefactos textuales. La figura 10.7(b) muestra una instancia del esquema representado en la figura 10.7(a), en la que la transformación modelo a modelo se ha implementado usando QVT Operational Mapping, y la transformación modelo a texto se ha implementado con Acceleo.

10.3.4 Otras transformaciones con propósito de validación

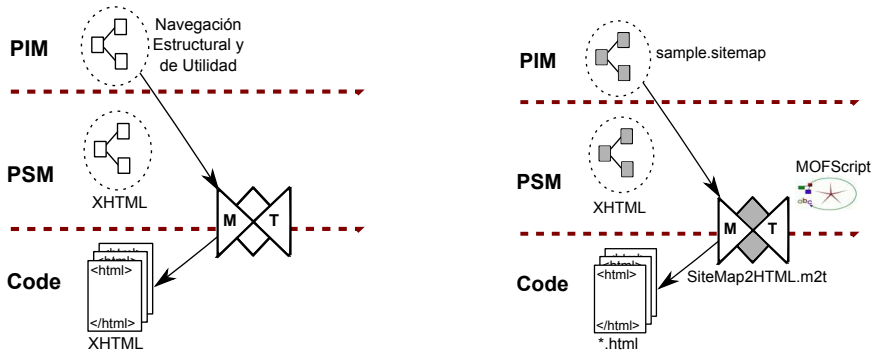
El último grupo de transformaciones está compuesto por aquéllas que se usan con propósito de validación, es decir, su objetivo es generar artefactos que tienen propósitos de validación. En la figura 10.8 se muestra un ejemplo de este tipo de transformación. La figura 10.8(a) muestra un esquema en el que mediante una transformación modelo a texto, se obtienen una serie de archivos `HTML` que se usan para que el usuario tenga un prototipo rápido de qué aspecto tendrá la navegación de utilidad y estructural del sitio. Note que en este caso, no se pasa por los modelos específicos de plataforma con objeto de obtener más rápidamente el bosquejo `HTML` de la navegación estructural y de utilidad. Detalles más concretos de este tipo de transformación se pueden obtener en [321] y en la sección 8.2.3 del capítulo 8. Una instancia de este esquema en la que la transformación modelo a texto se ha implementado en MOFScript, como se muestra en la figura 10.8(b).



(a) Esquema

(b) Ejemplo de instancia del esquema

Figura 10.7: Esquema de composición simétrica



(a) Esquema

(b) Ejemplo de instancia del esquema

Figura 10.8: Esquema de transformación de validación

10.4 Las estrategias de transformación y/o composición MWACSL aplicadas a la navegación

Esta sección profundiza en cómo se aplican las estrategias de composición y transformación especificadas en la sección anterior al aspecto de navegación usando casos concretos y dando detalles de la implementación de estas transformaciones. En la subsección 10.4.1 se detalla la transformación directa implementada para obtener flujos de Spring Webflow a partir de modelos de flujo de navegación. La subsección 10.4.2 explica cómo se usan los modelos de anotación para añadir información de la interfaz gráfica a los modelos de flujo de navegación controlada. La subsección 10.4.3 se centra en la combinación de la navegación estructural y de utilidad, los flujos de navegación no con-

trolada y la interfaz gráfica. Finalmente, las transformaciones de validación aplicada a la navegación estructural y de utilidad se detallaron en la subsección 8.2.3, dejando este apartado para los detalles de las estrategias en las que están involucradas la composición de concerns.

10.4.1 De SWC a Spring Web Flow

En este caso estamos ante una transformación "sencilla", en el sentido de que la representación de los modelos SWC no difieren en demasía de los modelos de Spring Web Flow, ya que Spring Web Flow también modela los flujos de navegación controlada mediante estados. La figura 10.9 muestra los artefactos producidos como resultado de la implementación de esta estrategia. A nivel de PIM, el metamodelo de los flujos de navegación controlada (`SWC.ecore`) se ha implementado en Ecore, igual que el metamodelo de Spring Web Flow (`SpringWF.ecore`). La transformación de modelo a modelo (`SWC2springWF`) se ha implementado con QVT Operational Mapping y la modelo a texto (`SWF2XML.m2t`) con MOFScript.

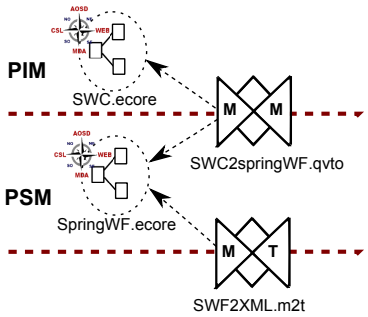
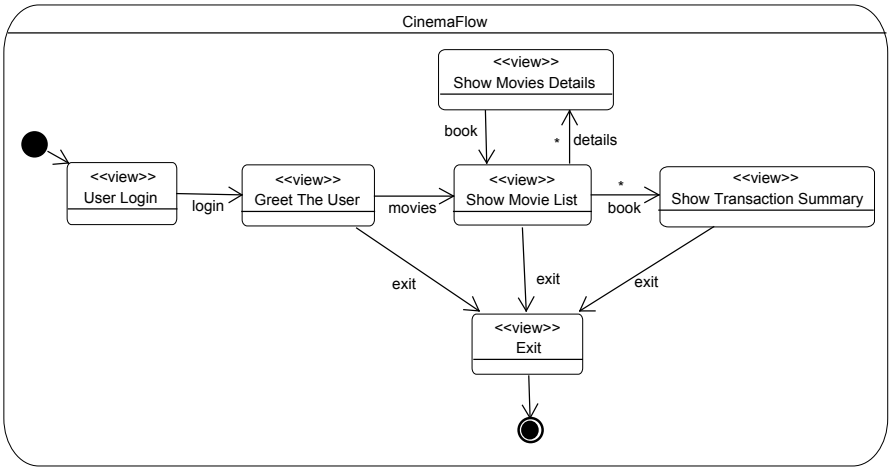


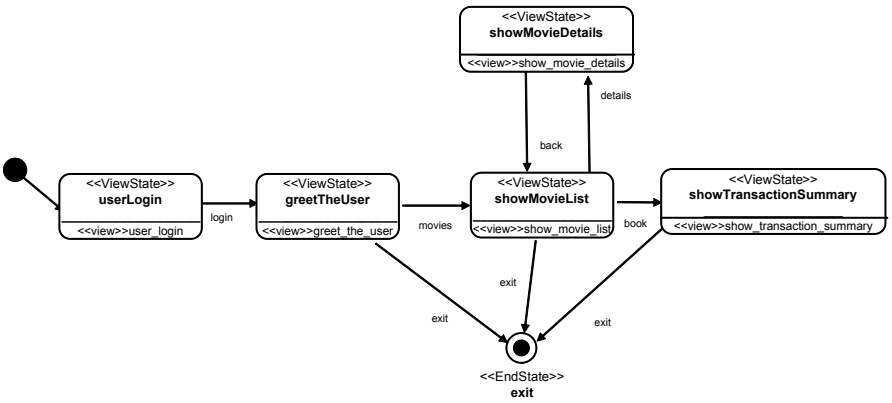
Figura 10.9: Artefactos involucrados en la transformación

En la figura 10.10 se puede ver un ejemplo de un modelo de entrada para la transformación y el modelo de salida resultado de esa transformación. El ejemplo es una adaptación del presentado en [209] en el que se muestra un diálogo para realizar la reserva de entradas para un espectáculo de forma telemática. El proceso de transformación consiste en lo siguiente: por cada flujo de navegación se genera un flujo de Spring Web Flow, por cada estado de tipo `FrontEndView` se genera un estado de tipo `ViewState` de Spring Web Flow (con una excepción que veremos un poco más adelante), y por cada transición de SWC se genera una transición en el diagrama de SWF.

Una de las cuestiones a tener en cuenta en la transformación a la hora de generar las transiciones en el modelo destino, es que hay que asegurarse de que previamente se hayan generado los estados origen y destino. La otra cuestión tiene que ver con la generación de los estados finales. En SWC los estados finales no llevan asociada ninguna vista, sin embargo, en Spring Web Flow sí. Esto implica que los estados finales de Spring Web Flow se corresponden con la secuencia formada por un estado de tipo



(a) Modelo original.



(b) Modelo transformado

Figura 10.10: Modelo original y modelo transformado

`FrontEndView` del que parte una transición que conduce a un estado final en SWC. Si se fija en la figura 10.10(a), este caso lo puede encontrar en el estado `exit`, que no es generado como `ViewState` sino como `EndState`. Esta es la única excepción a la regla anterior.

Un resumen de las correspondencias o *mappings* definidas en QVT Operational Mappings para implementar esta transformación se puede ver en la figura 10.11 y la tabla 10.1. Mientras que en la figura se representan las correspondencias usando la sintaxis concreta de los lenguajes SWC y SWF y tomando como base el ejemplo de la figura 10.10, la tabla muestra un resumen de estas correspondencias. Si el lector quiere más detalles acerca de estas transformaciones, puede descargar su código fuente de: `http://www.lsi.us.es/~reinaqu/org.mwacsl/html/`.

Lo más destacable de la correspondencias y que puede resultar más curioso al lector al echarle un vistazo a la tabla 10.1, son las dos siguientes cuestiones:

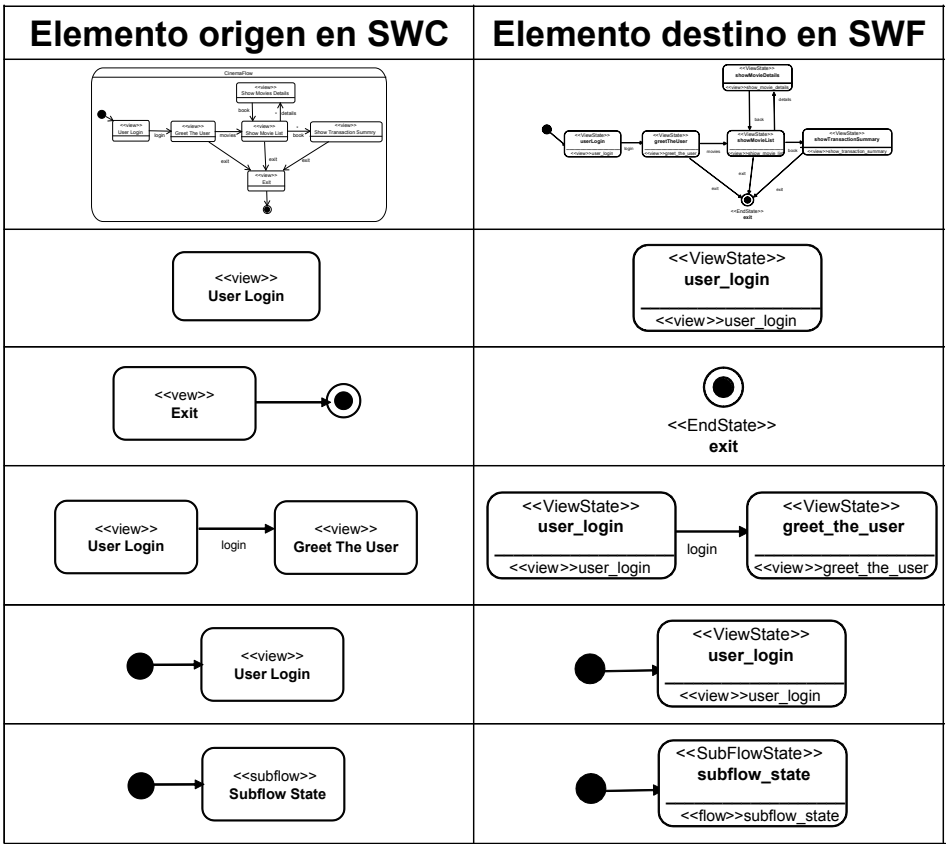


Figura 10.11: Correspondencias entre elementos de SWC y SWF representadas mediante la sintaxis concreta

Metaclase Origen	Metaclase Destino	Mapping
WebFlow	Flow	toSpringWF()
FrontEndView	ViewState	toViewState()
State	EndState	toEndState()
Transition	StateTransition	toTransition()
NavigationMachine	Flow	toSpringWF()

Tabla 10.1: Resumen de correspondencias en `SWC2SWF`

- La correspondencia `toEndState` relaciona un estado (metaclase `SWC::State`) con un estado final (metaclase `SWF::EndState`). Esta generalización a la hora de definir la correspondencia se ha definido así por cuestiones de implementación, ya que lo que correspondería exactamente con la metaclase `SWF::State` sería el patrón formado por un estado de tipo `SWC::FrontEndView` con una transición que conduce a un pseudo-estado de tipo `EndState`.
- La correspondencia entre los elementos que marcan el inicio de los flujos no es

directa (comprobará que en la última línea de la tabla la metaclase origen es `NavigationMachine` y la destino `Flow`). Esto se debe a que el elemento de la sintaxis concreta que marca el inicio del flujo no está recogido en ninguna metaclase concreta, sino que es una relación (una referencia) entre las metaclases `WebFlow` y `State`. De esta forma, la marca de inicio de flujo se genera en la correspondencia entre los dos flujos.

10.4.2 De SWC a JSP

SWC se ha diseñado de forma que está centrado en los flujos de navegación y, por tanto, no tiene ningún detalle acerca de cómo se representarán los elementos de navegación en la interfaz. Por ejemplo, una transición de SWC puede representarse gráficamente como un botón, como una imagen, como una etiqueta (la propia de los enlaces web). Para poder generar una aplicación web, es necesario incorporar este tipo de información al modelo de flujo de navegación.

Con objeto de mantener los modelos de interfaz y de navegación separados, en `MWACSL` se añade esta información mediante modelos de anotación, que son un tipo de modelo de *weaving*. En este caso concreto, el modelo de anotación propuesto, que se ha denominado *modelo de anotación GUI*, se ha definido como una extensión al modelo de anotación propuesto en [83] (figura 10.4). Los artefactos involucrados en la composición de estos modelos se representan en la figura 10.12, en la que se puede comprobar cómo las entradas al proceso de transformación y composición son el metamodelo de flujo de navegación (`SWC.ecore`), el metamodelo de anotación de interfaz gráfico (GUI) (`mwGUIannotation.ecore`), y un metamodelo de `JSP` (`JSP.ecore`). Como se puede observar en la figura, los metamodelos de SWC y de anotación de interfaz, se han definido dentro del marco de `MWACSL`, mientras que el metamodelo de `JSP` se ha tomado del componente `JSP` de la versión 0.9 de MoDisco [167]. MoDisco [125] es un *framework* extensible para desarrollar herramientas dirigidas por modelos que den soporte a la modernización de sistemas existentes vía ingeniería inversa.

La decisión de usar el metamodelo de `JSP` del proyecto MoDisco se ha tomado para promover la reutilización, y sacar así partido de los componentes ya desarrollados en MoDisco. De esta forma, una vez obtenido el modelo (`.jspxmi`) se puede utilizar el componente `JSP Generation` de MoDisco para generar el código `JSP`, ya que este componente permite, a través de transformaciones de modelo a texto definidas mediante una plantilla Acceleo/MTL [125], generar un conjunto de ficheros `JSP` a partir de un modelo (`.jspxmi`).

El metamodelo de anotación de interfaz, es propio de `MWACSL`, aunque está basado en el propuesto en [83](figura 10.4). Una de las ventajas de extender este metamodelo es que se pueden usar las herramientas desarrolladas en la plataforma AMMA para definir modelos de *weaving*, con lo que nos ahorramos crear un editor.

Una representación gráfica y más intuitiva del mecanismo de anotación se muestra en la figura 10.13. Note que esta representación tiene solo fines didácticos, y que las

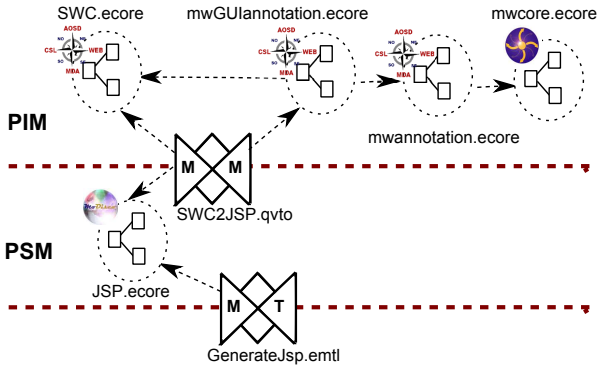


Figura 10.12: Esquema de composición y transformación de modelos mediante modelos de anotación

anotaciones se definen en un modelo separado. En ella puede ver como a cada transición se le asocia un elemento gráfico. En concreto, las transiciones `login`, `exit1`, `exit2`, `exit3`, `back`, `details` y `book` tienen asociado un botón, mientras que la transición `movies` tiene asociado un enlace, es decir, se representará como una etiqueta textual.

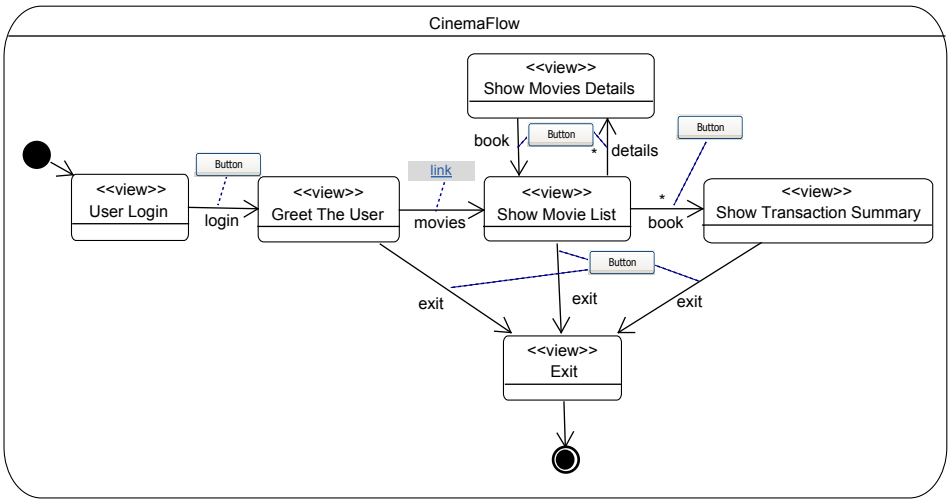


Figura 10.13: Esquema de flujo con Información de interfaz gráfica

Como se ha comentado anteriormente, el metamodelo de anotación propuesto en [83] y representado en la figura 10.4, se ha tenido que adaptar en nuestra propuesta. Dicha adaptación se ha resaltado con una elipse roja en la figura 10.14, y consiste en cambiar la cardinalidad de la relación `annotatedModelElement`. En la propuesta original, la cardinalidad es `0..1`, lo que indica que una anotación se asocia estrictamente a un elemento del modelo anotado o bien al modelo en su conjunto (caso de cardinalidad 0). En nuestro caso, hemos relajado esta restricción cambiando la cardinalidad a "*",

con objeto de que la misma anotación se pueda aplicar a varios elementos del modelo anotado.

En la figura 10.14, además de la adaptación, se muestra la extensión realizada al modelo de anotación. Las anotaciones están inspiradas en el cartucho para JSF de AndroMDA [15], ya que en este cartucho se modelan las diferentes formas de representar un enlace en una interfaz gráfica de cualquier sitio web (como enlace, propiamente dicho, como botón o como imagen). La extensión del metamodelo está formada por el conjunto de metaclases que en la figura 10.14 aparecen representadas con fondo gris, y que se detallan a continuación:

GUIAnnotation. Representa una anotación de interfaz gráfica. Extiende a la metaclase `Annotation` del metamodelo de anotación.

LinkAnnotation. Representa una anotación que se deberá hacer sobre un elemento que represente un enlace. En el caso de SWC, este elemento será una transición.

Button. Representa una anotación que indica que un enlace se mostrará como un botón.

Link. Representa una anotación que indica que un enlace se mostrará como hipervínculo. Tiene un atributo que representa la `URL` de la raíz que da acceso al sitio.

Image. Representa una anotación que indica que un enlace se mostrará como una imagen. Tiene un atributo que representa la `URL` de la raíz que da acceso al sitio. Y otro que representa la `URL` del archivo de imagen.

La necesidad de definir los atributos para las metaclases `Link` e `Image` se presenta en la figura 10.15. En esta figura se muestra el trozo de código `HTML` que habría que generar para cada una de la distintas representaciones gráficas de un enlace. En un óvalo se ha marcado la parte de código `HTML` que varía con los distintos modelos. La parte común a las tres subfiguras (10.15(a),10.15(b) y 10.15(c)) es el nombre del enlace (`login` en la subfigura 10.15(a) y `exit` en la subfiguras 10.15(b) y 10.15(c)). Esta parte común se obtiene del modelo referenciado, que en el caso de SWC será el nombre de la transición.

De la parte no común, hay una información que comparten el enlace y la imagen, `cinema_service.htm`, que se corresponde con la `URL` de entrada al sitio, y que da lugar al atributo `urlContainer` que aparece en las metaclases `Link` e `Image`. Mientras, que la imagen, además, necesita la `URL` para localizar el archivo de imagen.

Aunque existe toda una infraestructura para implementar *weaving* de modelos en la plataforma AMMA, está diseñada para trabajar con transformaciones implementadas en `ATL`. Como en `MWACSL` las transformaciones se han implementado con QVT Operational Mapping, se ha tenido que crear toda la infraestructura desde cero y se ha usado el mecanismo de `QVT` denominado *QVT/Blackbox* para invocar métodos de utilidad para

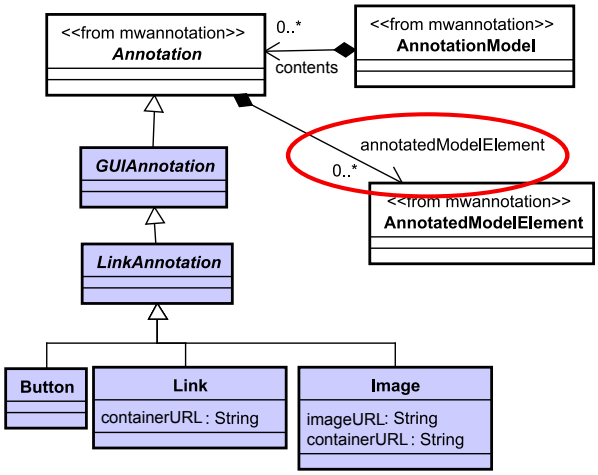
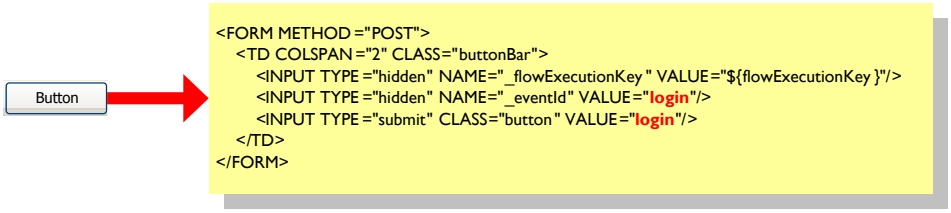
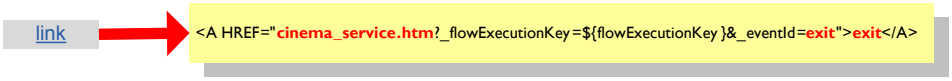


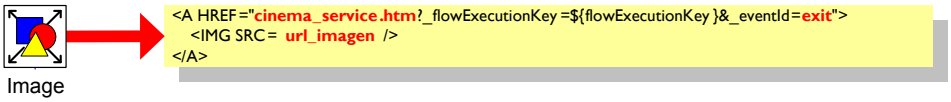
Figura 10.14: Extensión de metamodelo de anotación para las anotaciones de GUI



(a) HTML generado para un botón.



(b) HTML generado para un enlace



(c) HTML generado para una imagen

Figura 10.15: Trozo de código HTML a partir de los distintos elementos de la interfaz gráfica con los que se relaciona una transición de SWC

el proceso transformación escritos en otros lenguajes, Java en nuestro caso. Un poco más adelante se darán detalles de para qué ha sido necesario utilizar este mecanismo.

El resumen de las correspondencias implementadas se muestra en la tabla 10.2. Básicamente, un modelo de SWC se corresponde con un modelo `JSP`, cada estado de tipo `FrontEndView` se corresponde con una página `JSP` y cada transición con un enlace. Aunque por el resumen, la transformación pueda parecer sencilla, puesto que solamente tiene tres correspondencias, lo que predomina es la parte operacional, ya que dentro

de la página JSP a generar hay que crear diferentes elementos que se corresponden con elementos JSP y HTML.

Metaclase Origen	Metaclase Destino	Mapping
WebFlow	Model	toModel()
FrontEndView	Page	toPage()
Transition	Element	toLink()

Tabla 10.2: Resumen de correspondencias en SWC2JSP

El modelo de anotación se incorpora en la correspondencia `toLink`, tal y como se muestra en el listado 10.1. Las consultas `isShownAsButton`, `isShownAsLink` e `isShownAsImage` acceden al modelo de anotación para comprobar si la anotación es un botón, un enlace o una imagen, respectivamente. En caso de que no sea de ninguno de los tipos anteriores, la transición generará los elementos correspondientes a un botón, que es la opción por defecto.

Listado 10.1: Correspondencia toLink()

```
1 mapping swcmm::Transition::toLink(parent: xmlmm::Element):xmlmm::
    Element{
2
3   if (self.isShownAsButton()) then{
4     self.createLinkButton(result,parent);
5   }else{
6     if (self.isShownAsLink())then{
7       self.createLinkLink(result, parent);
8     }else{
9       if (self.isShownAsImage())then{
10        self.createLinkImage(result, parent);
11      }else{
12        self.createLinkButton(result,parent);
13      }endif;
14    }endif;
15  }endif;
16
17 }
```

Un ejemplo de estas consultas se muestra en el listado 10.2. En esta consulta, en primer lugar se obtienen todos los objetos del modelo de anotación que sean de tipo `Button`, es decir, todas las anotaciones de tipo botón (almacenado en la variable `buttons`). En segundo lugar, obtiene el conjunto de todos los elementos que tienen una anotación (variable `anotModElRef`). Luego, de los elementos anotados se queda con aquéllos cuya referencia es `ref`. Note que `ref` es la forma de enlazar los elementos del modelo de anotación con los elementos del modelo de flujo. La referencia usada es el fragmento de `URI` que también se usa en la serialización `XMI` del modelo y se calcula mediante la llamada a `getURIFragment`. Se ha tomado este fragmento de `URI` porque

es la que usa el editor de AMW. Finalmente, la transición deberá representarse como un botón si en el conjunto de elementos asociados con la anotación de tipo `Button` se incluye la transición que sirve de contexto a la consulta.

Listado 10.2: Consulta isShownAsButton

```
query swcmm::Transition::isShownAsButton():Boolean{
var buttons: Set(amwAnot::Button):=
       anotModel.objects()[amwAnot::Button];
var anotModElRef: Set(amwAnot::AnnotatedModelElementRef):=
             anotModel.objects()[amwAnot::AnnotatedModelElementRef];

var ref:String := self.oclAsType(ecore::EObject).getURIFragment();
var ame:Set(amwAnot::AnnotatedModelElementRef):=anotModElRef->
             select(amr |amr.ref = ref);

return buttons->collect(b | b._end.element)->includesAll(ame);
}
```

El método `getURIFragment` usado en la consulta del listado 10.2 se ha definido en Java mediante el mecanismo QVT/Blackbox, tal y como se muestra en el listado 10.3. La anotación Java `@Operation(contextual=true)` indica que el primer parámetro del método será el contexto de la operación cuando se llame desde el código `QVT`. En este caso, el contexto sería `EObject`.

Listado 10.3: Consulta Java creada usando el mecanismo QVT/Blackbox

```
@Operation(contextual=true)
public static String getURIFragment (EObject eObject) {
  Resource xmiResource = eObject.eResource();
  String rdo = null;
  if (xmiResource != null){
    XMLResource xmlr =(XMLResource) xmiResource;
    rdo = xmlr.getURIFragment(eObject);
  }
  return rdo;
}
```

10.4.3 De SWC y navegación estructural y de utilidad a JSP

Los flujos de navegación y la navegación estructural y de utilidad se combinan usando un esquema simétrico, tal y como se muestra en la figura 10.7. La información de cómo se enlazan los elementos de los flujos y la navegación estructural se incorpora mediante un modelo de *weaving*. Este modelo de *weaving* se ha definido como una extensión al modelo básico de AMW mostrado en la figura 10.3. Los puntos de extensión son las metaclases `WModel`, `WLink` y `WLinkEnd`. El resultado de la composición es un modelo conforme al metamodelo de `JSP` definido en el proyecto Modisco (Ver sección 9.4). Hay

que tener en cuenta que este tipo de composición solamente se realizará en flujos de navegación que no son controlados, ya que normalmente en los flujos de navegación controlada se eliminan de la interfaz todos los enlaces, excepto los propios del flujo, para evitar que el usuario siga un camino distinto al marcado por el flujo. La figura 10.16 muestra los artefactos involucrados en la composición. Los metamodelos de flujos de navegación y navegación estructural y de utilidad se han implementado en Ecore, y se encuentran en los archivos `SWC.ecore` y `sitemap.ecore`, respectivamente. El metamodelo de anotación se ha definido en el archivo `mwnavigation.ecore` como una extensión del metamodelo básico de *weaving* (en `mwcore.ecore`). Estos tres artefactos, junto con el metamodelo de `JSP` (`JSP.ecore`), sirven como entrada a la transformación de modelo a modelo definida en QVT Operational Mapping `SWCSitema2JSP.qvto`. El propio metamodelo de `JSP.ecore` sirve como entrada a la transformación modelo a texto `GenerateJsp.emtl` que se encarga de generar los archivos `JSP`.

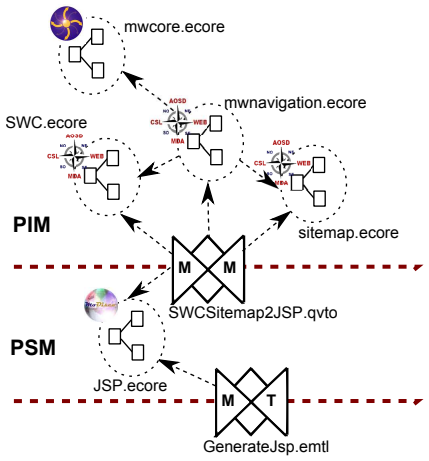


Figura 10.16: Artefactos involucrados en la composición

La figura 10.17 muestra la extensión realizada al metamodelo básico de *weaving*. cuyas metaclases están estereotipadas con `<<mwcore>>`, mientras que las metaclases que componen la extensión se han coloreado en gris. Las metaclases que componen este metamodelo son las siguientes:

NavigationWModel. Es la clase raíz del metamodelo de *weaving*, y, por lo tanto, extiende a `WModel`. Un modelo de *weaving* de navegación relaciona exactamente a un modelo de navegación estructural y un modelo de flujo, lo que se modela mediante las relaciones de composición `structuralNavM` y `navFlowM` con `WModelRef`, que representa una referencia a un modelo. Además, un modelo de *weaving* de navegación está compuesto por una serie de localizaciones (`Location`).

Location. Es una metaclase abstracta que representa un enlace entre un nodo de un mapa del sitio y un flujo o un estado, es decir, indica dentro del mapa del si-

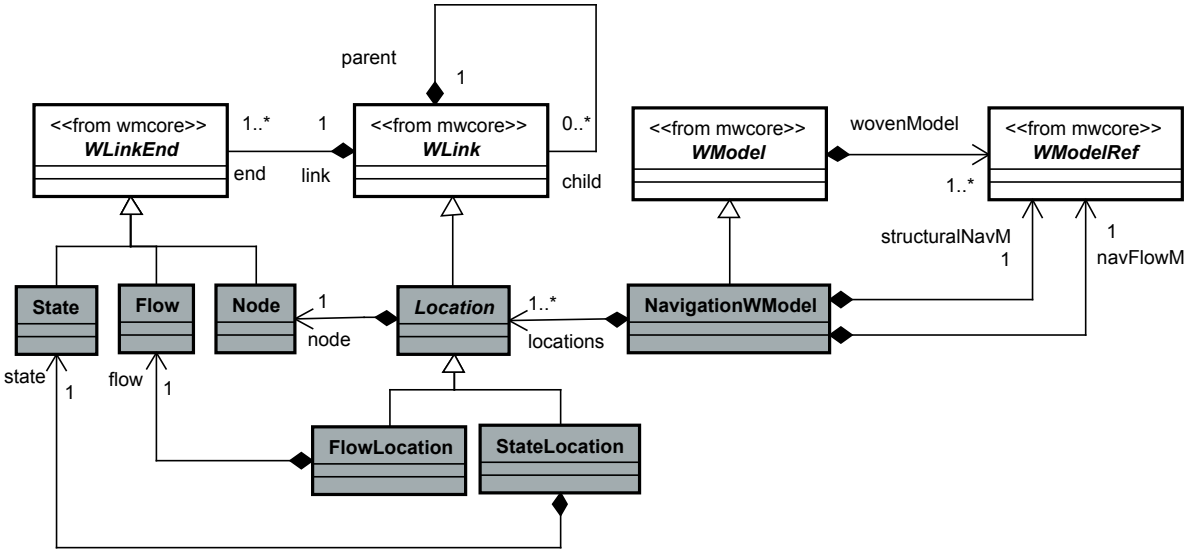


Figura 10.17: Extensión de metamodelo weaving para componer la navegación estructural y de utilidad y los flujos de navegación

tio dónde está localizado el flujo o un estado del flujo. Como una localización representa un enlace, la metaclase `Location` extiende a `WLink`.

FlowLocation. Extiende a `Location` y representa la localización del flujo en el mapa del sitio. El localizar un flujo en un nodo implica que el estado inicial del flujo estará localizado en ese nodo.

StateLocation. Extiende a `Location` y representa la localización de un estado en el mapa del sitio.

State. Extiende a `WLinkEnd` y representa un estado de un flujo de navegación.

Flow. Extiende a `WLinkEnd` y representa un flujo de navegación.

Node. Extiende a `WLinkEnd` y representa un nodo de un mapa del sitio.

10.5 Sumario

Este capítulo se ha centrado en ver cómo se relacionan los distintos tipos de conceptos en `MWACSL`. Los conceptos se modelan mediante lenguajes específicos de dominio, así que, la relación entre los mismos se reduce a una relación entre modelos. Una primera aproximación a la relación entre modelos y la transformación directa entre SWC y Spring Web Flow se ha publicado en [336]. La transformación de modelo a modelo con propósitos de validación que obtiene un modelo `HTML` a partir de un modelo de

navegación estructural y de utilidad se publicó en [329]. La transformación modelo a texto equivalente se publicó en [328].

The true method of knowledge is experiment.

William Blake

11 Validación

En este capítulo se presentan los casos de estudio utilizados para probar la propuesta realizada en esta tesis. Para ello, el capítulo se ha estructurado de la siguiente forma: en la sección 11.2 se presenta un ejemplo sencillo de flujo de navegación. En la sección 11.3 se muestra un ejemplo sencillo de mapa de sitio web. La sección 11.4 se dedica a TDG Scholar, un motor de búsqueda de bibliografía científica del área de informática. El último caso de estudio es el conocido sitio de comercio electrónico Amazon.com. Tras presentar los casos de estudio se hace un análisis sobre la cómo se relacionan los concerns *navegacionales estudiados en esta tesis (sección 11.6). El capítulo acaba con un sumario.*

11.1 Introducción

Los casos de estudio utilizados para probar el *framework* propuesto en esta tesis se presentan de forma progresiva, en el sentido de que primero se muestran casos de estudio sencillos, en los que se modelan los aspectos navegacionales de forma aislada, para continuar con casos de estudio más complejos, a medida que se avanza en el capítulo. El primer ejemplo que se presenta, en la sección 11.2, es un flujo sencillo en el que se solicita al usuario la introducción de sus datos personales y los datos de un pedido para realizar una compra. En este ejemplo sencillo se muestra el flujo de navegación aislado de cualquier otro *concern* navegacional. El ejemplo presentado en la sección 11.3 es un caso de estudio centrado en la navegación estructural y de utilidad, como en la sección anterior, de forma aislada. En este caso, se trata de un sitio web sencillo formado por páginas estáticas que recogen la información relacionada con

las tareas docentes e investigadoras de un profesor. El primer caso de estudio es TDG Scholar (`http://scholar.tdg-seville.info/`), un motor de búsqueda de bibliografía científica del área de la informática. El segundo caso de estudio es el conocido sitio de comercio electrónico Amazon.com (`http://www.amazon.com`). En cada caso de estudio se hace un análisis de la navegación estructural y de utilidad y/o de los flujos web más interesantes desde el punto de vista didáctico.

Según Wieringa y Heerkens [407] existen distintas formas de evaluar las tecnologías aplicadas en la ingeniería de requisitos, que ordenadas según el grado de validez externa que proporcionan son: modelado, experimentos de campo o de laboratorio, investigación en acción, proyecto piloto y casos de estudio. De acuerdo a esta clasificación, la validación del trabajo presentado en esta tesis, y mostrado en este capítulo está en el nivel de modelado usando dos ejemplos reales (`TDG-Scholar` y `Amazon`).

11.2 Un ejemplo sencillo de flujo de navegación

El primer caso de estudio es un flujo sencillo para una aplicación simple de comercio electrónico. El flujo se puede ver en la figura 11.1. En ella se representa un diálogo mediante el que se solicita al usuario de la aplicación sus datos personales y los datos del pedido a realizar. En la figura se puede observar cómo lo primero que se encuentra el usuario es una página en la que hay un enlace que permite iniciar el diálogo. Una vez pulsado este enlace aparece un formulario en el cual el usuario tendrá que introducir sus datos personales. Después, tras pulsar en el botón `Go ahead!` del formulario, el usuario tendrá que introducir los datos del producto a solicitar, para poder finalmente, completar el pedido. Las flechas que unen una página con otra muestran los posibles movimientos del usuario. Hay que destacar que de ninguna forma se puede acceder a un paso intermedio del diálogo, a no ser que se haya pasado por los anteriores. Es decir, para llegar al formulario de introducción de los datos del pedido, previamente se ha tenido que pasar por el formulario de introducción de datos personales.

Este mismo flujo especificado mediante la sintaxis concreta para Spring Web Flow presentada en el capítulo 9 se muestra en la figura 11.2. Así, en primer lugar se utiliza un estado de tipo `<<ViewState>>` para modelar el hecho de presentarle al usuario un formulario en el que se solicitan sus datos personales. La acción de renderizado asociada al este estado se encarga de inicializar el formulario, mientras que con el estereotipo `<<view>>` se indica qué página es la encargada de mostrar el formulario.

La acción asociada a la transición `submit` se encarga de validar los datos que el usuario ha introducido y de ligarla a los objetos que replican el formulario en el servidor. Mediante el estado de tipo vista llamado `orderDetailsView` se realiza la misma operación, solo que en este caso, en el formulario se solicitan los datos del producto que vaya a adquirir el cliente. La acción asociada a la transición `buy` se encarga de validar los datos introducidos por el usuario. Por último, dado que al usuario se le da la opción de cancelar los datos del pedido, mediante un estado de decisión se encamina el flujo,

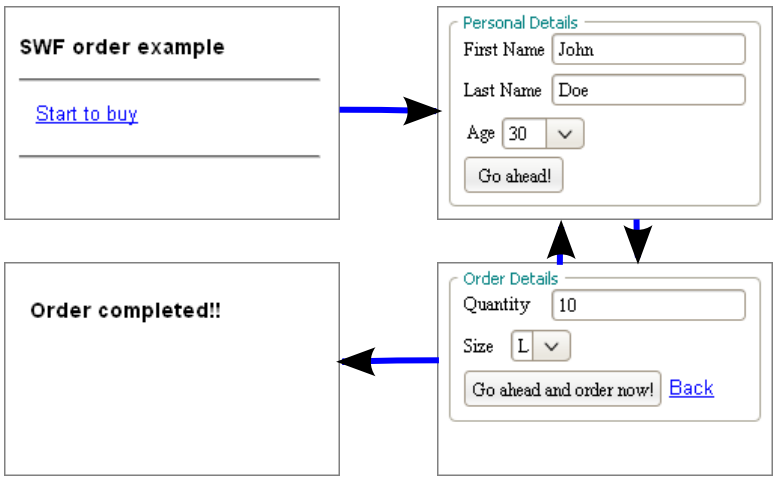


Figura 11.1: Un flujo de navegación controlado sencillo para una aplicación de comercio electrónico

bien hasta su estado final, bien hacia una nueva solicitud de los datos del pedido.

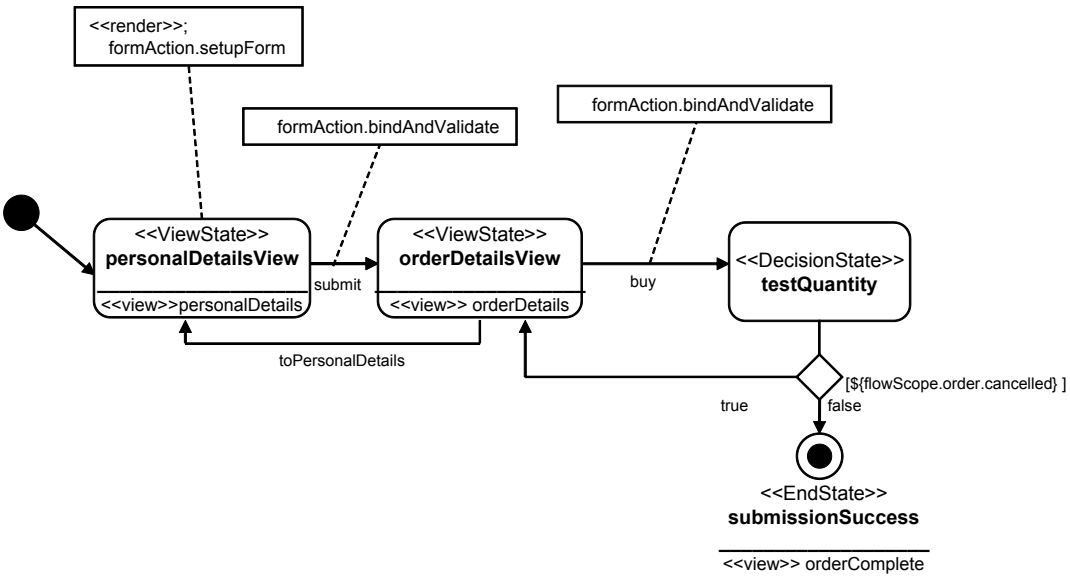


Figura 11.2: Flujo simple para una aplicación de comercio electrónico

El resultado de la aplicación de las transformaciones modelo a texto presentadas en la sección 9.3.3 al modelo de la figura 11.2 es el archivo `XML` representado en la figura 11.3.

```
<?xml version="1.0" encoding="UTF-8"?>
   <flow xmlns="http://www.springframework.org/schema/webflow"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="
             http://www.springframework.org/schema/webflow
             http://www.springframework.org/schema/webflow/spring-webflow-1.0.xsd">


     <start-state idref="personalDetailsView"/>

     <view-state id="personalDetailsView" view="personalDetails">
            <render-actions>
                   <action bean="formAction" method="setupForm"/>
            </render-actions>

            <transition on="submit" to="orderDetailsView">
                   <action bean="formAction" method="bindAndValidate">
                          <attribute name="validatorMethod" value="validatePersonalDetails"/>
                   </action>
            </transition>
     </view-state>


     <view-state id="orderDetailsView" view="orderDetails">
            <transition on="buy" to="testQuantity">
                   <action bean="formAction" method="bindAndValidate">
                          <attribute name="validatorMethod" value="validateOrderDetails"/>
                   </action>
            </transition>
            <transition on="toPersonalDetails" to="personalDetailsView"/>
     </view-state>

     <decision-state id="testQuantity">
            <if test="${flowScope.order.cancelled}"
                   then="orderDetailsView"
                   else="submissionSuccess"/>
     </decision-state>

     <end-state id="submissionSuccess" view="orderComplete"/>
</flow>
```

Figura 11.3: Archivo XML generado mediante las transformaciones para el flujo simple para la aplicación de comercio electrónico

11.3 Sitio web de un investigador

El segundo caso de estudio se centra en la navegación estructural y de utilidad y corresponde a un sitio web sencillo para gestionar la actividad docente e investigadora de un docente. La navegación estructural y de utilidad de este sitio web se puede ver en la figura 11.4, en la que se muestra una distribución horizontal tanto de la navegación principal como de la navegación local, situada debajo de ésta. La navegación de utilidad se sitúa por encima de la principal, en la parte superior derecha.

Para especificar la navegación estructural y de utilidad se ha utilizado el metamodelo y la sintaxis concreta introducidos en la sección 8.2. El mapa del sitio web del investigador es sencillo, tal y como se puede observar en la figura 11.5. Sin embargo, en él se pueden observar, de un vistazo, los tipos de navegación (estructural y de utilidad)

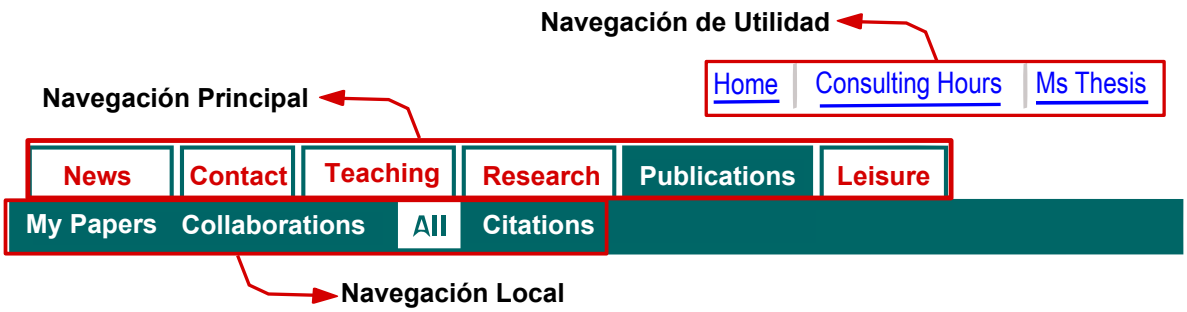


Figura 11.4: Navegación estructural y de utilidad del sitio web de un investigador

que se especifican mediante el lenguaje de modelado mencionado anteriormente. La navegación de utilidad se sitúa fuera de la jerarquía del sitio, la navegación principal cuelga del nodo raíz del mapa del sitio y se convertirá en un conjunto de enlaces que aparecerán en todas las páginas del sitio web. La navegación local en este mapa del sitio se sitúa por debajo del nodo `5-Publications`, y se traduce en un conjunto de enlaces que solamente aparecerán en las páginas situadas bajo el nodo de publicaciones.

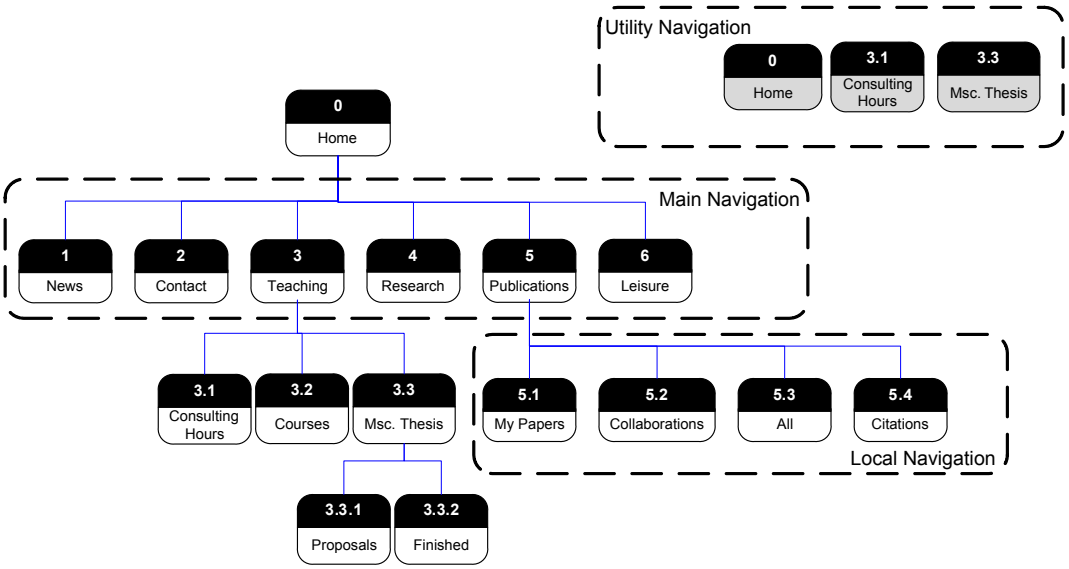


Figura 11.5: Un ejemplo de mapa del sitio web

Como ya se adelantó en el capítulo 8, el mapa especificado en la figura 11.5, se ha utilizado como entrada para las transformaciones modelo a texto presentadas en la sección 8.2.3. El resultado de estas transformaciones es un conjunto de archivos `HTML` estáticos, uno por cada uno de los nodos representados en la figura 11.5.

11.4 TDG Scholar

TDG Scholar [161] es un motor de búsqueda de bibliografía científica dentro del área de informática. En su versión 2.0, las principales funcionalidades que proporciona son la de búsqueda y la que en el sitio web se conoce como `Binder`, que no es más que una forma de almacenar aquellas publicaciones en las que el usuario está interesado. A estas funcionalidades se puede acceder a través de la navegación principal del sitio que se muestra en la parte superior derecha, tal y como puede verse en la figura 11.6. En la misma figura se puede ver la navegación de utilidad, situada al mismo nivel que la navegación principal, pero en la parte superior izquierda, en lugar de la derecha.

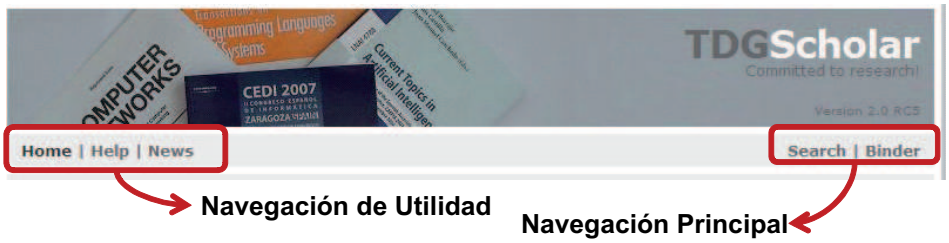


Figura 11.6: Navegación principal y de utilidad en TDG Scholar Version 2.0

TDG Scholar es una aplicación web que se sigue desarrollando, de forma que se irán ampliando las funcionalidades que proporciona. Así, un esbozo de la navegación principal que le aparecerá a un usuario autentificado es la que aparece en la figura 11.7. En este caso, TDG Scholar no tiene definida navegación local, solamente navegación principal y de utilidad.

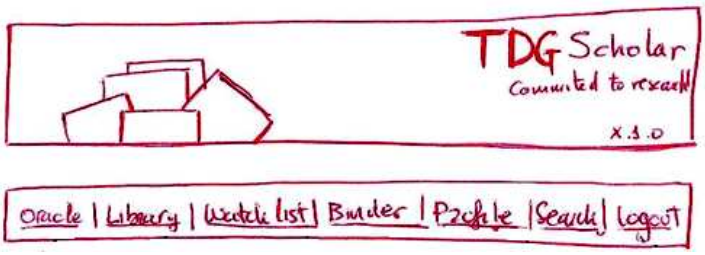


Figura 11.7: Extracto de wireframe representando la navegación principal de TDG Scholar

Con respecto a los flujos de navegación, TDG Scholar, al ser un motor de búsqueda, no presenta ningún flujo de navegación controlada. De hecho, la mayoría de los diálogos se centran en dos modelos de interfaces gráficas distintas, como se detallará un poco más adelante. En las dos subsecciones siguientes se muestran los modelos del mapa del sitio

y de flujos especificados para este sitio web, conforme a los metamodelos especificados en el capítulo 8.

11.4.1 Sitemap de TDG Scholar

El *sitemap* de TDG Scholar se puede ver en la figura 11.8. En TDG Scholar no hay navegación local. La navegación principal que se representa en el mapa del sitio es la que se le mostrará a un usuario que previamente se ha autentificado. La posibilidad de registrarse no está disponible en la versión 2.0 de TDG Scholar, aunque está prevista para versiones futuras. De esta forma, el mapa del sitio de la figura 11.8 corresponde al diseño del *wireframe* de la figura 11.7.

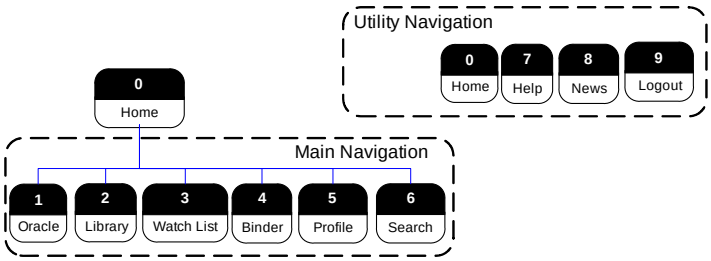


Figura 11.8: Sitemap con la navegación principal y de utilidad de TDG Scholar

11.4.2 Flujos web en TDG Scholar

TDG Scholar consta básicamente de dos diseños de pantalla: la pantalla inicial de bienvenida o *Home* (figura 11.9), y la pantalla de búsqueda o *Search* (figura 11.10). La pantalla mediante la que se gestiona el archivador o *Binder* es muy similar a la de búsqueda, así que nos centraremos en esta última. Una interacción básica de un usuario con TDG Scholar consistirá en introducir una consulta (utilizando la interfaz de la figura 11.9) y TDG Scholar mostrará los resultados tal y como se presenta en la figura 11.10. Se puede decir que esta interacción del usuario no representa un flujo de navegación controlado, ya que no se restringe el movimiento del usuario por la aplicación.

La figura 11.11 muestra un modelo basado en diagramas de estados que representa la interacción del usuario con TDG Scholar a la hora de realizar una búsqueda. Como se puede ver en la figura, hay varios estados marcados con el estereotipo `<<view>>`. Cada uno de estos estados representa a una vista distinta que se le presenta al usuario. El estado `enterSearchCriteria` representa la vista *Home* de la figura 11.9. Centrándonos exclusivamente en el proceso de búsqueda, desde este estado, el usuario puede realizar una búsqueda pulsando el botón que aparece con el icono de una lupa y que está situado a la derecha del cuadro de texto. La transición etiquetada como `search` que parte de este estado conduce al estado `viewResults`, que muestra el resultado de la búsqueda.

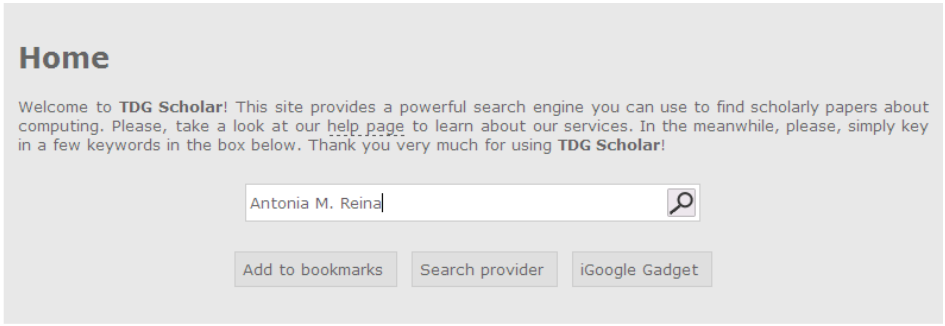


Figura 11.9: Pantalla de inicio de búsqueda

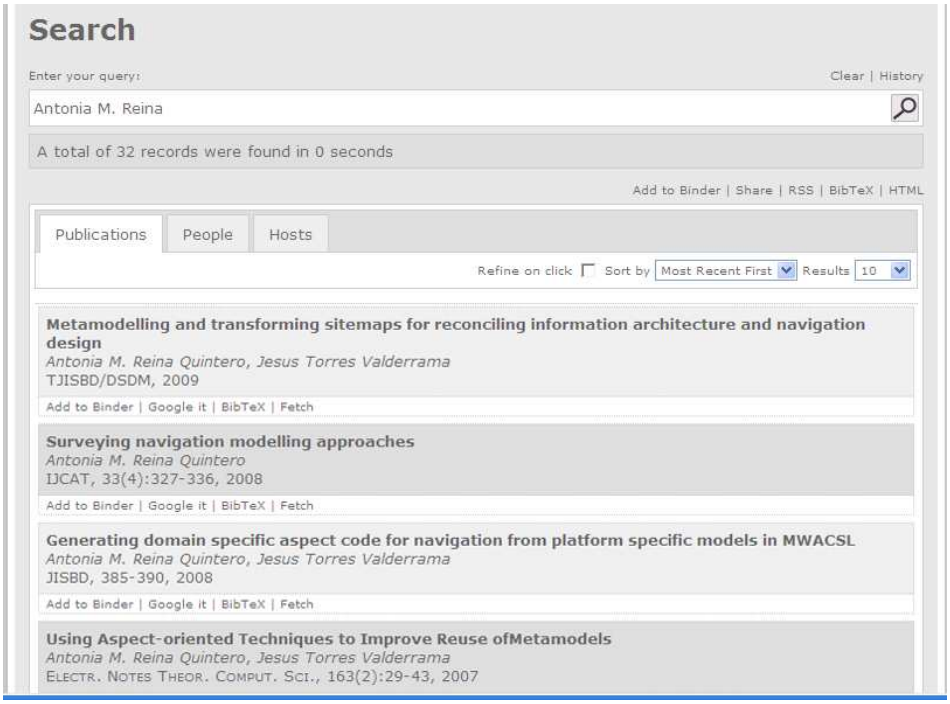


Figura 11.10: Pantalla de resultado de búsqueda

Desde esta vista se puede acceder al historial de búsquedas realizadas (mediante la transición `history`). Esta transición se corresponde con el enlace `history` representado en la figura 11.10. Igualmente, hay transiciones que conducen a vistas con las publicaciones en formato `RSS` (`getRSS`), en formato `HTML` (`export HTML`), como archivo BibTex de bibliografía para latex (`export BibTex`) o compartir la consulta en alguna red social (`shareQuery`).

El estado `view Results` es un estado compuesto que contiene tres estados: `view`

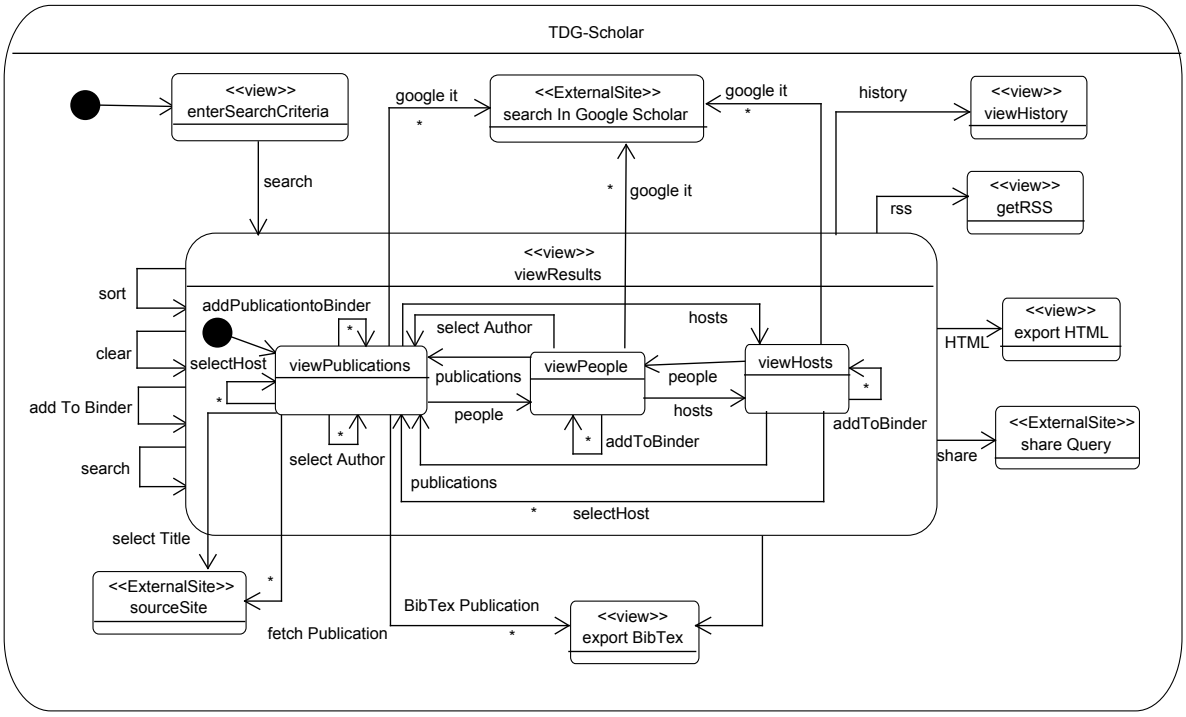


Figura 11.11: Diálogo Search de TDG-Scholar 2.0

`Publications`, `view People` y `view Hosts`. Estos estados no están marcados con el estereotipo `<<view>>`, porque no representan una vista independiente. Como se puede comprobar en la figura 11.10, estos estados se han implementado mediante una serie de pestañas, que representan tres categorías distintas de navegación: publicaciones, autores y lugares en los que se publica. Mientras que cuando está seleccionada la pestaña `Publications` como resultado de la búsqueda se muestra un conjunto de publicaciones, cuando se selecciona `People` se muestra el conjunto de autores de esas publicaciones. Finalmente, al tener seleccionada la pestaña `Hosts` se muestran los sitios en los que se han publicado los artículos. Realmente, el conjunto de resultados de la búsqueda es el mismo en los tres casos, la diferencia está en cómo se agrupa ese conjunto de resultados.

Otro estereotipo que aparece en el modelo es `<<ExternalSite>>`, que indica que se está accediendo a un sitio externo. Así, desde los estados `view Publications`, `view People` y `view Hosts` se puede realizar una búsqueda en Google Scholar mediante las transiciones etiquetadas como `google it`. También se puede acceder al sitio original de la publicación (`sourceSite`), bien pulsando en el título de la publicación (transición `selectTitle`), bien a través de un enlace `Fetch` (transición `fetch Publication`).

Finalmente, hay que destacar que hay una serie de transiciones cuyo estado origen y destino es el mismo, como consecuencia de que no cambian la vista que se muestra al usuario. Así, por ejemplo, la transición `add To Binder` tiene como consecuencia añadir

una publicación al archivador (o *Binder*). El aspecto visual de la vista cambia, puesto que se resaltan en amarillo aquellas publicaciones que se han añadido al archivador, sin embargo, esto no se refleja en el modelo de flujo de navegación, ya que la vista a la que conduce sigue siendo la misma.

11.5 Amazon.com

Amazon.com es uno de los primeros y más exitosos sitios que aparecieron en internet dedicado al comercio electrónico. Aunque en sus comienzos Amazon.com se dedicó a la venta de libros, con el tiempo la compañía ha ido diversificando su negocio y actualmente ofrece diferentes líneas de productos, que van desde CD's y DVD's hasta ropa, muebles y comida.

Al igual que se ha ido expandiendo el negocio, el sitio web dedicado al comercio electrónico que le da soporte también ha ido evolucionando. Con la expansión de la compañía el sitio web tuvo que evolucionar, y uno de los principales problemas a los que se tuvieron que enfrentar los diseñadores fue la escalabilidad. Para resolverlo, los diseñadores optaron por definir la navegación principal del sitio mediante un menú desplegable, de forma que el añadir items al menú no interfiera con el diseño del resto de la página. Esto se consigue haciendo que este menú aparezca por defecto replegado, y dejando al propio usuario la opción de expandirlo. En la figura 11.12 se muestran: el menú colapsado que contiene la navegación principal (en la parte izquierda), la navegación local que se ha distribuido horizontalmente en la parte superior de la página, y la navegación de utilidad, situada también horizontalmente por encima de la navegación principal y la local.

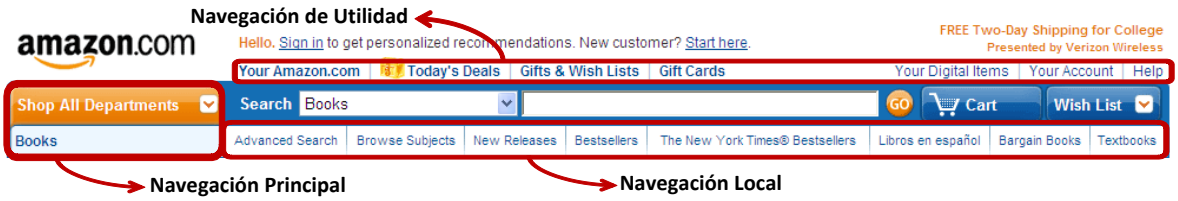


Figura 11.12: Navegación estructural y de utilidad en Amazon.com

Si vemos el menú de navegación principal desplegado (figura 11.13), podemos decir que la distribución de la navegación es una mezcla entre dos distribuciones comunes, la distribución en L-invertida y la distribución de la navegación con menús desplegables. En la figura 11.13 también se puede comprobar que el menú de navegación principal se ha definido en dos niveles, de forma que el primer nivel sirve para estructurar la navegación, mientras que los enlaces del segundo nivel son realmente los que conducen a las secciones concretas del sitio.

Figura 11.13: Menú desplegable que representa la navegación principal en Amazon.com

Con respecto a los flujos de navegación, esta aplicación es también más compleja que las de las secciones anteriores, ya que se mezclan diálogos de navegación controlada con navegación libre. Así, en el proceso de compra se mezclan los dos tipos de interacciones. Cuando el usuario realiza una compra, desde la aplicación se le obliga a dar una serie de pasos, llegando incluso a desaparecer la navegación principal y local, para que no se desvíe del camino trazado por los diseñadores de la aplicación. De esta forma, para realizar una compra, el usuario se tiene que autentificar, seleccionar la dirección a la que se le envía el pedido y escoger las opciones de envío y el modo de pago, todo esto antes de realizarlo, estando este diálogo dentro de un flujo controlado. Por otra parte, cuando el usuario olvida su clave, comienza un diálogo que deja a un lado los pasos controlados del flujo de compra, volviendo a aparecer el menú de navegación principal.

11.5.1 Sitemap de Amazon

El mapa del sitio de Amazon.com es un poco más complejo que lo visto hasta ahora. La figura 11.14 muestra la porción del mapa del sitio de Amazon.com que especifica la navegación principal. Como puede comprobarse, la navegación principal se ha definido en dos niveles. Esto se debe a que la navegación principal se ha diseñado como un menú desplegable de dos niveles, que, por defecto, aparece plegado (figura 11.12) y es el propio usuario el encargado de desplegarlo. El único caso en el que el menú aparece desplegado es en la página raíz del sitio, aunque en este punto solamente se despliegan los items situados en el primer nivel de la jerarquía.

Si observamos el mapa del sitio de Amazon.com, también comprobaremos que solamente los nodos situados en el segundo nivel de la navegación principal conducen a secciones del sitio. Los nodos situados a primer nivel solamente sirven para estructurar lo que en el sitio web acabará representado como un menú.

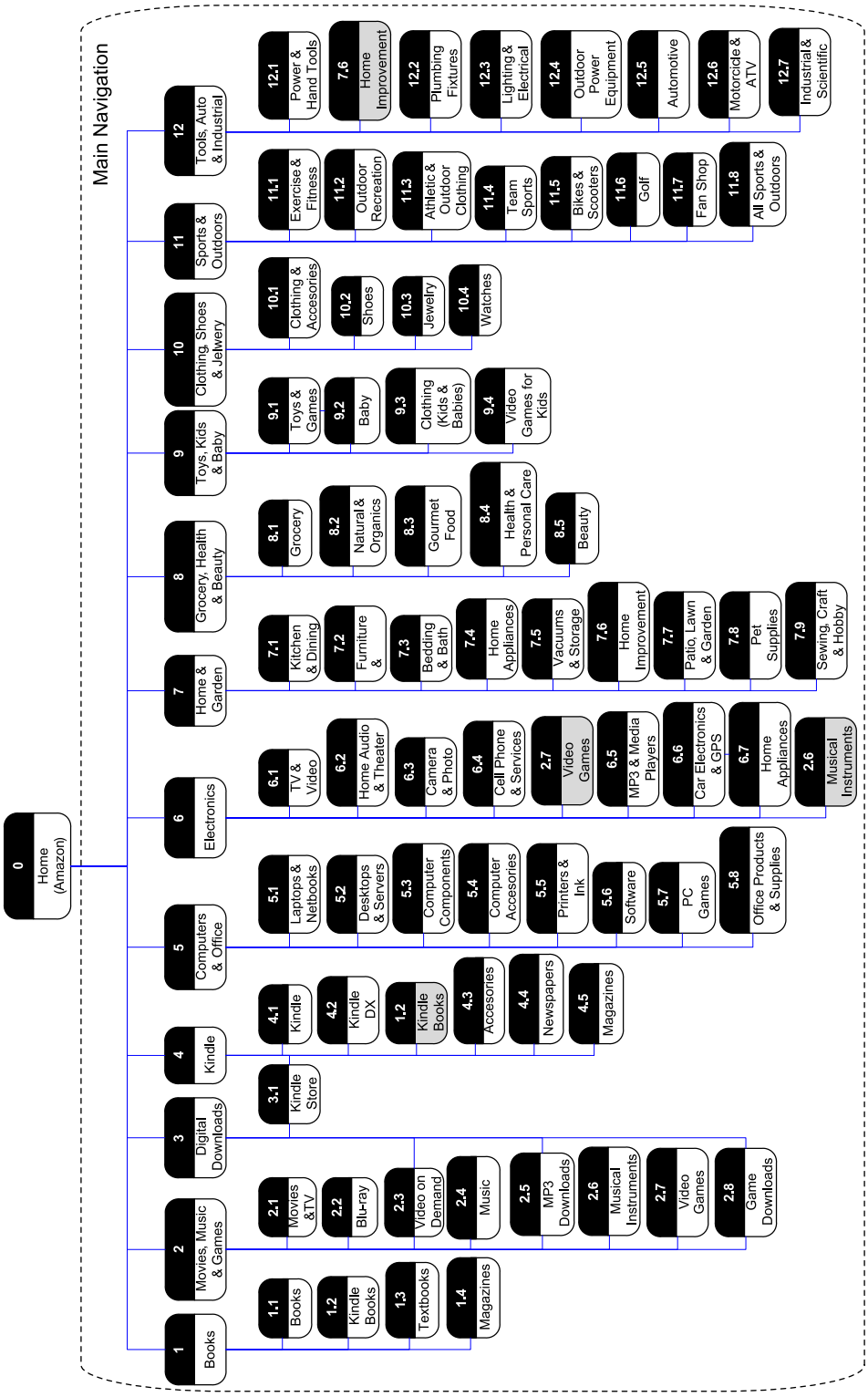


Figura 11.14: Extracto de sitemap que muestra la navegación principal en Amazon.com

Otra cuestión interesante que se plantea en este sitio web es el hecho de que hay partes del sitio a las que se puede acceder utilizando distintas rutas del menú de navegación principal. Por ejemplo, a la sección `Video on Demand` se puede acceder tanto desde el item `Movies, Music & Games` como desde `Digital Downloads`. Para modelar esta circunstancia, se dejan dos opciones alternativas, aunque con una semántica equivalente. La primera opción, es la opción por defecto, y consiste en dibujar dos conexiones, una entre los nodos `2` y `2.3`, y otra entre los nodos `3` y `2.3`. Los nodos de tipo referencia se representan con fondo gris. Un ejemplo de referencia es el nodo `1.2-Kindle Books` de la figura 11.14. Recuerde que las referencias se definen con objeto de no enmarañar el mapa del sitio. Así, el nodo `1.2-Kindle Books` aparece dos veces en el mapa del sitio, una colgando de `1-Books` y otra, colgando de `4-Kindle`. La única diferencia es que la que cuelga de `4-Kindle` aparece dibujada con el fondo en gris, como forma de resaltar que ese nodo ya ha aparecido antes.

La navegación local también es más compleja, llegando incluso a existir varios niveles de navegación local (figura 11.15). Así, por ejemplo, bajo el nodo `1.2-Kindle Books` existe un nivel del navegación local, y uno de los nodos situados en este área, el etiquetado como `1.2.3-Discussions` también tiene su propia área de navegación local, de forma que cuando el usuario entra en el nodo `1.2.3`, el área marcada en la figura 11.12 como navegación local cambia su contenido y pasa a mostrar enlaces a los nodos `1.2.3.1`, `1.2.3.2`, `1.2.3.3`, `1.2.3.4`, `1.2.3.5` y `1.2.3.6`. En aras de mejorar la legibilidad de los diagramas, la navegación principal y la navegación local se han presentado de forma separada, aunque, obviamente, están conectadas en el mismo mapa del sitio. Igualmente, existen más áreas de navegación local, bajo los nodos `1.1`, `1.3`, `1.4`, `2.1`, `2.2` y `2.3`, que no se muestran en el capítulo, porque presentan una problemática similar. Si el lector está interesado puede ver el mapa completo del sitio en `http://www.lsi.us.es/~reinaqu/org.mwacsl/html/`.

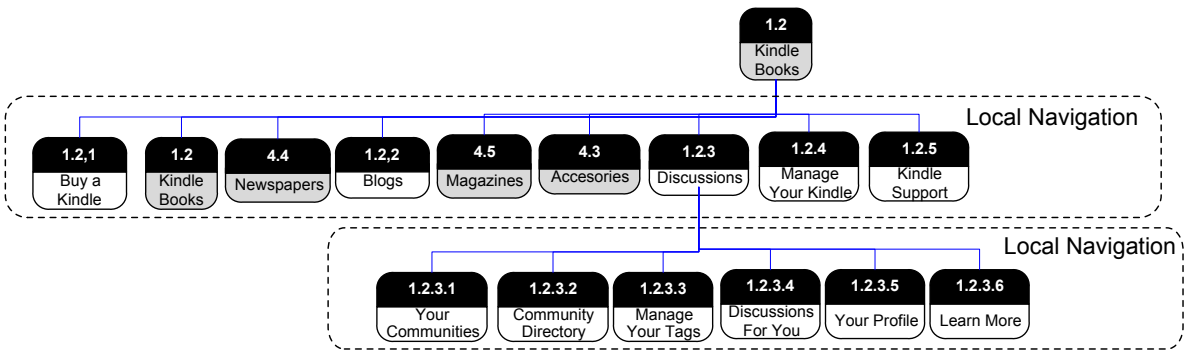


Figura 11.15: Extracto de sitemap que muestra parte de la navegación local en Amazon.com

Finalmente, el extracto del mapa del sitio que representa la navegación de utilidad en Amazon.com se representa en la figura 11.16. En este caso, a diferencia de la navegación

estructural, la navegación de utilidad se sitúa fuera de la jerarquía del sitio.

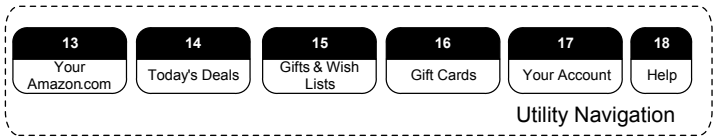


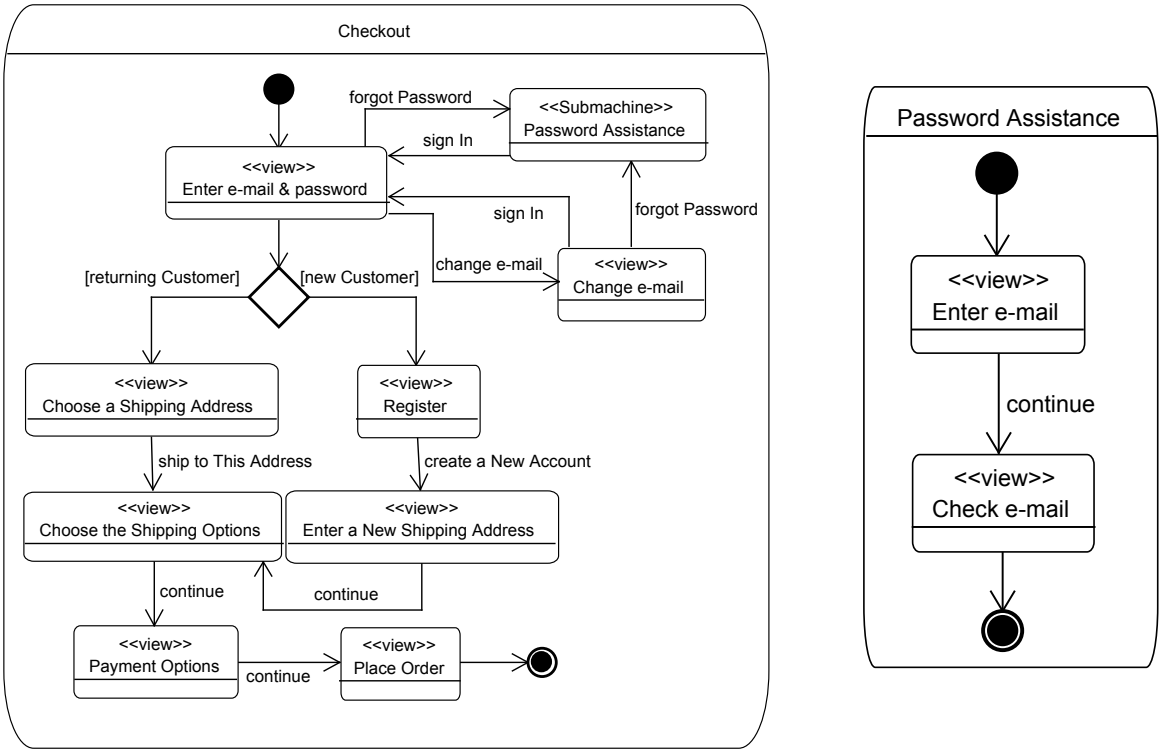
Figura 11.16: Extracto de sitemap que muestra la navegación de utilidad en Amazon.com

11.5.2 Flujos web en Amazon.com

Como ya se ha comentado en la introducción de esta sección, uno de los flujos fundamentales en Amazon es el de *checkout*, ya que la compra es uno de los objetivos básicos del sitio. El flujo de compra se ha modelado en nuestro *framework* tal y como se muestra en la figura 11.17(a). El estado inicial del flujo es un estado en el que se le solicita al usuario su dirección de correo electrónico y su clave, con objeto de identificarle. Desde este estado un usuario puede seguir adelante con el proceso de compra, siempre que se haya autentificado correctamente. Si hay algún error en el proceso de autentificación (como por ejemplo que la dirección de correo introducida no esté registrada, o la clave no sea la correcta), se vuelve al mismo estado. Esto se refleja en el modelo mediante la transición `error` que parte del estado `Enter e-mail and password` hacia si mismo. En este estado también se le da la opción al usuario de modificar la dirección de correo electrónico con la que se registró en la aplicación (transición `change Email`) o pedir ayuda si es que no recuerda la clave que utilizó en su registro (transición `forgot Password`). Como se puede observar en la figura, el estado al que se llega cuando se pide ayuda con la clave es de tipo `<<SubFlow>>`. Este tipo de estado no solamente tiene sentido para simplificar los modelos, sino que en el caso de Amazon, cuando el usuario entra en el flujo de ayuda con la clave, abandona la navegación controlada que existía en el proceso de compra, volviendo incluso a aparecer el área de navegación principal y de utilidad en la página (figura 11.18).

Tras el estado `Enter e-mail and password` se produce una bifurcación del flujo de control, dependiendo de si el usuario es nuevo en la aplicación, en cuyo caso tendrá que registrarse, o es un usuario ya registrado. Un usuario nuevo tendrá que registrarse (estado `Register`) e introducir la dirección a la que se le debe enviar la mercancía (estado `Enter a new shipping address`). Un usuario ya registrado solamente tendrá que escoger la dirección de envío (estado `Choose a shipping address`). Tras estos pasos, los dos tipos de usuarios confluyen en el estado `Choose the shipping options`, donde escogerán las opciones de envío. Finalmente tendrán que elegir la opción de pago (estado `Payment Options`) y realizar el pedido (`Place Order`).

En la figura 11.17(b) se muestra cómo se ha modelado el subflujo para el diálogo de asistencia para la clave (`Password Assistance`). En este caso el diálogo está formado



(a) Modelo de flujo de checkout en Amazon

(b) Subflujo password assistance en Amazon

Figura 11.17: Diálogo para realizar una compra en Amazon

solamente por dos estados, un primer estado en el que se le solicita al usuario su dirección de correo electrónico, y segundo estado que le muestra al usuario un mensaje avisándole de que revise su cuenta de correo electrónico.

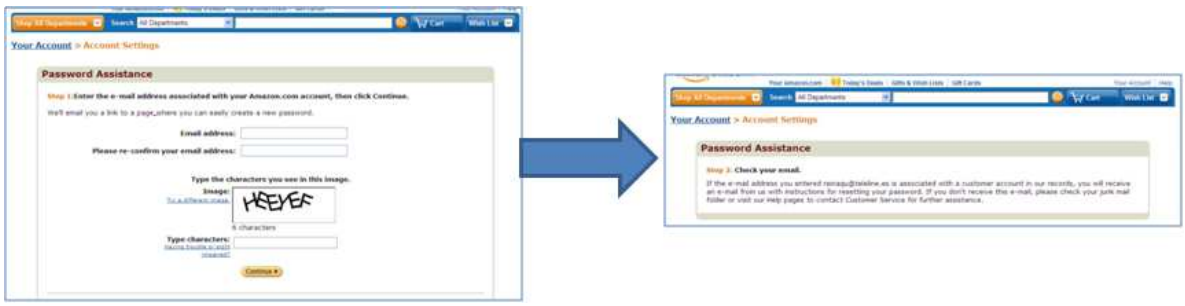


Figura 11.18: Subflujo password assistance en Amazon

11.6 Interacción entre concerns

Una vez que se han presentado los casos de estudio, en los cuales se han mostrado, por una parte, la navegación estructural y de utilidad, y por otra, los flujos de navegación, hay que tener en cuenta la interacción entre los mismos. En los casos de estudio, las interacciones entre *concerns* más interesantes son:

- Interacción entre concerns de navegación.

  Este tipo de interacción se presenta, sobre todo, en el sitio web de Amazon.com, que es el que tiene flujos de navegación controlados. La interacción de los flujos de navegación controlado con la navegación estructural y de utilidad hace que estos últimos desaparezcan. Así, la interacción entre estos dos conceptos navegacionales hace que varíe el comportamiento de uno de ellos, la navegación estructural y de utilidad, ya que se supone que debe aparecer en todas las páginas del sitio. En la figura 11.19 se muestra uno de los pasos del proceso de compra en el que se ve cómo se ha eliminado de la página la navegación principal y la de utilidad.

Figura 11.19: Paso del flujo checkout de navegación controlada de Amazon.com

- Interacción entre navegación y presentación.

  La interacción entre la navegación y la presentación está presente en dos de los casos de estudio (la web del investigador y TDG Scholar). Tiene lugar cuando se marca visualmente el enlace correspondiente al lugar en el que se encuentra el usuario. Así, en la figura 11.4 se puede ver cómo la pestaña `Publications` está marcada con colores distintos al resto de pestañas. También el ítem `All` correspondiente al área de navegación local se muestra en colores distintos al

resto. Se puede decir en este caso que el color en el que se dibuja un enlace depende del paso de navegación en el que se esté.

La misma situación se produce en TDG Scholar, marcando la pestaña correspondiente a la vista actual en un color distinto al del resto. En la figura 11.10 se puede comprobar como la pestaña `Publications` está resaltada con un color distinto al del resto. Igualmente, TDG Scholar marca el ítem de navegación global seleccionado poniendo en negrita el texto del enlace actualmente seleccionado (figura 11.20).

Figura 11.20: Ítem de la navegación principal resaltado en negrita de TDG Scholar

11.7 Sumario

En este capítulo se han presentado los casos de estudio utilizados para validar el trabajo presentado en esta tesis. Se ha empezado por un caso de estudio sencillo, cuyo foco son los flujos de navegación. Luego, se ha introducido otro caso de estudio en el que se ha aislado la navegación estructural y de utilidad. Estos dos primeros casos de estudio están se han definido con fines didácticos.

Finalmente, se han definido los flujos de navegación y la navegación estructural y de dos aplicaciones web: TDG Scholar y Amazon.com. Como colofón, se ha realizado un estudio de los tipos de interacción entre los conceptos y cómo se manifiestan éstas en los sitios usados como caso de estudio. El primer caso de estudio se introdujo en [334] y se refinó en [321]. El segundo caso de estudio se ha utilizado en las publicaciones [328] y [329].

Parte IV

Conclusiones y Trabajos Futuros

People do not like to think. If one thinks, one must reach conclusions. Conclusions are not always pleasant.

Helen Keller

12 Conclusiones y Trabajos Futuros

El objetivo de este capítulo es resumir, concluir y presentar las líneas futuras de trabajo resultado del trabajo de tesis presentado en este documento. Para ello, el capítulo se ha estructurado de la siguiente forma: en la sección 12.1 se resume y concluye la tesis, mientras que en la sección 12.2 se apuntan las líneas de trabajo futuro, las cuales se han clasificado por bloques de actuación.

12.1 Conclusiones

Nuestra hipótesis de partida era que los cambios en la navegación de las aplicaciones web son frecuentes y, por lo tanto, las empresas de desarrollo de software tienen un gran interés en reducir los costes derivados de los mismos. Planteamos como tesis al comienzo de este documento que es posible desarrollar una propuesta basada en modelos específicos de dominio a nivel independiente de la plataforma que separe los distintos *concerns* involucrados en la navegación.

Creemos que esta tesis ha quedado demostrada gracias a las contribuciones que hemos presentado, que son las siguientes:

- La concepción de la navegación desde la perspectiva de la orientación a aspectos, lo que desliga el diseño de la navegación del diseño del modelo conceptual.
- El considerar la ingeniería de modelos como una forma de implementar lenguajes específicos de *concerns*, especificando los lenguajes específicos de *concerns* en el espacio tecnológico de los modelos.

- La definición de un lenguaje de modelado específico de dominio para tratar y especificar la navegación estructural y de utilidad junto con una herramienta de modelado.

- La definición de un lenguaje de modelado específico de dominio para especificar la navegación controlada o flujos de navegación.

- La definición de un lenguaje de modelado específico de dominio para especificar flujos de navegación controlada en Spring Web Flow [89].

- La definición una serie de transformaciones usadas como prueba de concepto para generar la navegación estructural y de utilidad en `HTML`.

- La definición de una serie de modelos de *weaving* para combinar, por una parte, navegación e interfaz, y por otra, los propios *concerns* navegacionales.

Hemos evaluado nuestro enfoque mediante varios casos de estudio: un sitio web de un investigador, `TDG-Scholar` y `amazon.com`. Además, durante el desarrollo de esta tesis se han tenido que enfrentar una serie de retos, no solo conceptuales, a saber:

Combinar diferentes disciplinas El aclarar la terminología usada es una cuestión importante cuando se combinan diferentes disciplinas, ya que, el mismo término puede tener un significado distinto en distintas disciplinas. Un ejemplo es el término *weaving* que se usa tanto en la ingeniería de modelos como en la orientación a aspectos. Incluso el propio concepto de navegación puede ser percibido de forma distinta desde la arquitectura de la información y desde la ingeniería web.

Herramientas de implementación Aunque no es un reto conceptual, una de las cuestiones a las que nos hemos tenido que enfrentar durante el desarrollo de la tesis es lidiar con diferentes herramientas y tecnologías, sobre todo en el área del desarrollo del software dirigido por modelos, que aún no estaban maduras. Desde que se comenzó a trabajar en esta tesis, las herramientas han evolucionado mucho, aún así, creemos que la comunidad todavía tiene que trabajar más sobre todo en integrar herramientas.

12.2 Trabajos futuros

Los resultados presentados en esta tesis no son un punto y final, sino que se abren múltiples caminos en los que seguir profundizando. En los siguientes apartados apuntaremos algunas líneas para continuar el trabajo presentado en esta tesis, algunas de las cuales ya se han empezado a explorar.

12.2.1 Editores gráficos

Una de las discusiones que surgen con más frecuencia cuando se desarrolla un DSL es si los DSL's se deberían escribir mediante una notación gráfica o textual. En esta tesis se ha trabajado con la notación gráfica, y aunque el foco de la tesis no es la generación de editores, se ha empezado a trabajar con Eugenia [223, 126] y GMF [127] como herramientas para generar editores. En la figura 12.1 se muestra el editor desarrollado para mapas de sitio web.

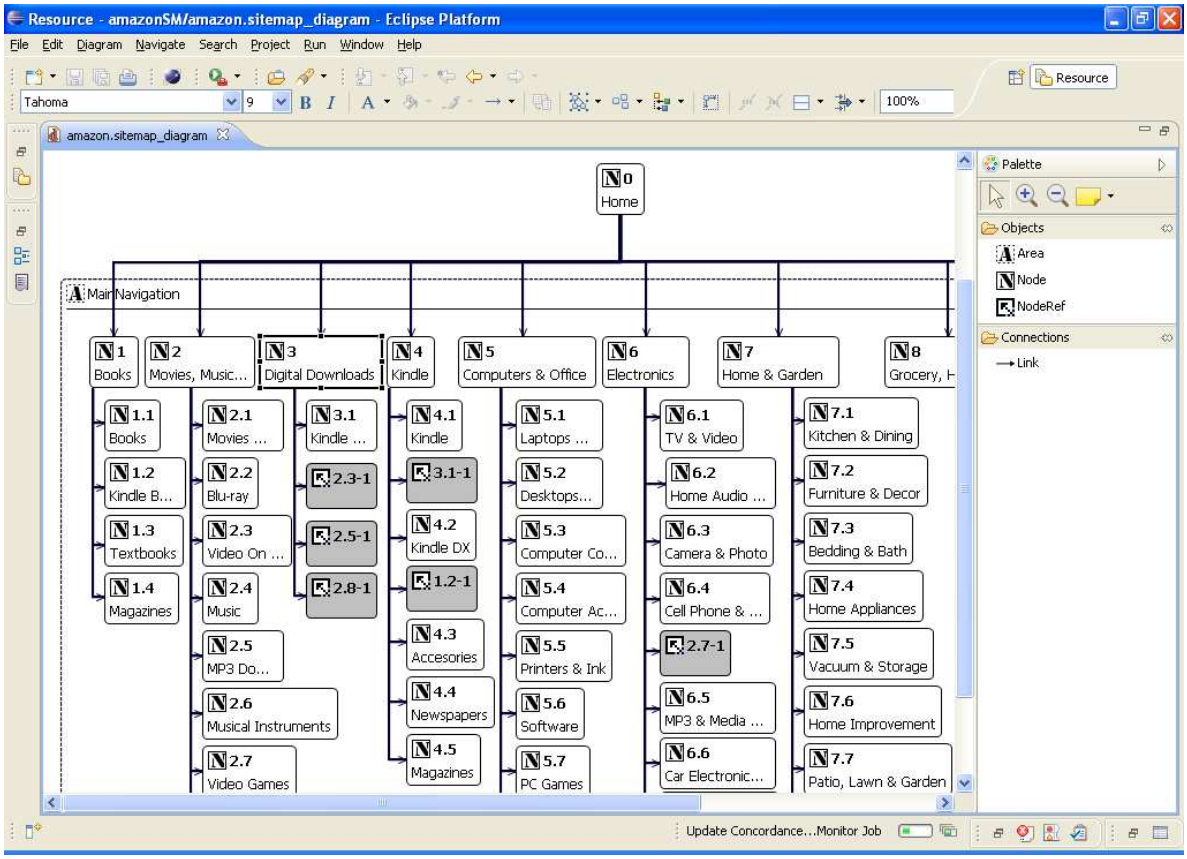


Figura 12.1: Captura de pantalla del editor de mapas de sitios web

En esta tesis se ha profundizado en cómo combinar los diferentes DSL's que representan distintos aspectos de un sistema. Una de las líneas de trabajo futura es cómo integrar los editores gráficos generados para cada uno de los aspectos del sistema en una sola herramienta, permitiendo generar editores para bancos de lenguajes.

12.2.2 Evolución de metamodelos y plataformas

Durante el desarrollo de la tesis se ha trabajado intensamente con diferentes metamodelos. Un inconveniente importante con el que se ha tenido que lidiar es que, normalmente,

un metamodelo se diseña de forma incremental, de manera que hasta llegar a una versión más o menos definitiva se pasa por muchas fases. Durante ese proceso, en algunas ocasiones, los cambios realizados hacen que los modelos que se habían definido para versiones anteriores ya no sean válidos para la nueva versión del metamodelo. En la línea de ayudar a adaptar los modelos y a las distintas versiones de los metamodelos por las que se vaya pasando, se tiene la publicación [327], en la que se hace una propuesta basada en orientación a aspectos para mejorar la reutilización de metamodelos. Además, no solamente existe la evolución propia de los metamodelos producida durante el desarrollo de los mismos, sino que, las plataformas concretas para las que se han definido metamodelos en el marco de la tesis, también evolucionan, lo que provoca que el metamodelo que representa la plataforma concreta tenga que modificarse para adecuarse a los cambios realizados en la misma. En esta cuestión también se apuntan ideas iniciales en el trabajo [335].

12.2.3 Otros CSLs y plataformas

Esta tesis se ha centrado en el aspecto de navegación, aunque nuestro objetivo es ir añadiéndole a `MWACSL` lenguajes específicos de dominio para trabajar con otros aspectos. El siguiente objetivo a corto plazo es separar *concerns* relacionados con la interfaz gráfica, tal y como la distribución de los elementos en la pantalla o *layout*. Para definir un `DSL` para el *layout* nos inspiramos en el *wireframe* [367], un entregable usado en el área de la arquitectura de la información. También queremos atacar otros aspectos tales como la persistencia.

En relación a las plataformas, nuestro objetivo es ir ampliando el abanico disponible y añadir nuevas plataformas como Seam [380] para navegación o Hibernate [374] para persistencia.

12.2.4 Modernización

La modernización de sistemas es un área de investigación que se dedica a reescribir o migrar un sistema heredado a tecnologías modernas. En esta tesis nos hemos centrado en realizar transformaciones de modelos de alto nivel hasta código, una línea de trabajo futuro es realizar las transformaciones inversas, en el sentido de generar modelos apropiados para las tecnologías con las que se trabaja en esta tesis a partir de código fuente. Como se ha presentado en los capítulos 9 y 10, ya se ha comenzado a trabajar con Modisco [125], el *framework* dentro del proyecto de modelado de Eclipse (EMP) [95], de tal forma que nuestro objetivo a más corto plazo en este área es la definición de un módulo de tipo `discoverer` que obtenga modelos SWF (capítulo 9) a partir de los archivos `XML` de definición de flujos.

Parte V

Apéndices

A

Listado Detallado de Publicaciones

En este apéndice se muestra un listado detallado de las publicaciones resultado del trabajo realizado en esta tesis. El conjunto de publicaciones se ha dividido en dos bloques: en primer lugar, se enumeran las publicaciones directamente relacionadas con esta tesis y en las que la autora aparece como primer autor; en segundo lugar, se enumeran aquellas publicaciones que no están directamente relacionadas con el trabajo de tesis pero que en cierto modo lo tocan tangencialmente y que son resultado de la colaboración con otros autores.

A.1 Publicaciones

Durante el desarrollo de esta esta tesis, la autora ha publicado varias partes de este trabajo en los siguientes artículos, ordenados por años:

2011

[ENVIADO A SAC'12] A. M. Reina, R. Corchuelo. On the asymmetric composition of controlled navigation flows and user interface concerns. *Enviado para publicación a la conferencia SAC'12.*

[ENVIADO A SAC'12] A. M. Reina, R. Corchuelo. A concern-specific language for dealing with structural and utility navigation. *Enviado para publicación a la conferencia SAC'12.*

2010

[JISBD-DSDM'10] A. M. Reina, J. Torres, M. Toro. De flujos de navegación a Spring Web Flow. Un primer acercamiento a las transformaciones verticales en MWACSL.

Publicado en las Actas de los Talleres de las Jornadas de Ingeniería del Software y Bases de Datos (TJISBD). 4(2), pages 19-28. ISSN: 1988-3455. SISTEDES, Septiembre, 2010.

[WEBIST'10] A. M. Reina, J. Torres. Sitemaps From A Model Driven Perspective. A First Step For Bridging The Gap Between Information Architecture And Navigation Design. *WEBIST 2010. Proceedings of the 6th International Conference on Web Information Systems and Technology*. Vol. 2, pages 111–117. ISBN: 978-989-674-025-2. INSTICC, Abril, 2010.

2009

[JISBD-DSDM'09] A. M. Reina, J. Torres. Metamodelling and transforming sitemaps for reconciling information architecture and navigation design. *Actas de los Talleres de las Jornadas de Ingeniería del Software y Bases de Datos (TJISBD)*. 3(2), pages 114–123. ISSN: 1988-3455. SISTEDES, Septiembre 2009.

2008

[IJCAT'08] A. M. Reina Quintero. Surveying navigation modelling approaches. *Int. J. Computer Application in Technology (IJCAT)*. 33(4), pages 327–336. ISSN: 0952-8091

[JISBD'08] A. M. Reina, M. Toro, J. Torres. Generating domain specific aspect code for navigation from platform specific models in MWACSL. In C. de la Riva J. Tuya A. Moreira, M. J. Suárez-Cabal, editor, *Publicado en las Actas de las XIII Jornadas de Ingeniería del Software y Bases de Datos (JISBD 08)*, pages 385–390. SISTEDES, October 2008.

2007

[JISBD-ZOCO'07] A. M. Reina, M. Toro, J. Torres. Integrando aspectos en MWACSL. In J. L. Álvarez, J. L. Arjona, R. Corchuelo, D. Ruiz, editors, *Publicado en las Actas de los Talleres de las Jornadas de Ingeniería del Software y Bases de Datos (TJISBD)*, Vol. 1.- No. 3, SISTEDES, 2007.

[JISBD-DSDM'07] A. M. Reina, J. Torres, M. Toro. El metamodelado de un framework: Spring Web Flow. In A. Vallecillo, V. Pelechano, A. Estévez, editors, *Publicado en las Actas de los Talleres de las Jornadas de Ingeniería del Software y Bases de Datos (TJISBD)*, Vol. 1.- No. 6, SISTEDES, 2007.

[ICWE-AEWSE'07] A. M. Reina, J. Torres, M. Toro. Improving the adaptation of web applications to different versions of software with MDA. In Marco Brambilla, Emilia Mendes, editors, *Proceedings of the Workshop on Adaptation and Evolution in Web Systems Engineering (AEWSE07). Published as: 7th International Conference on Web Engineering. Workshop Proceedings*, pages 101–107, 2007.

[ENTCS'07] A. M. Reina, J. Torres. Using aspect-oriented techniques to improve the reuse of metamodels. *Proceedings of the Second International Workshop on Aspect-Based and Model-Based Separation of Concerns in Software Systems (ABMB 2006)*, ENTCS 163(2):29–43, April 2007.

2006

[JISBD-DSOA'06] A. M. Reina, J. Torres, M. Toro. Hacia lenguajes de metamodelado orientados a aspectos. *Proceedings of the Workshop of Aspect-Oriented Software Development (DSOA'06). Collocated to XI Jornadas de Ingeniería del Software y Bases de Datos (JISBD'06)*. Sitges, Spain. October, 2006. Publicado por la Universidad de Extremadura como Informe Técnico TR-24/06. pp. 19–26

[AOSD-SE'06] A. M. Reina. Separation of Concerns and Model Driven Development Applied to Web Systems. *AOSD'06 Student Extravaganza: Spring School. Held in conjunction with the Aspect-Oriented Software Development Conference (AOSD'06)*. Bonn, Germany. March, 2006.

[AOSD-PS'06] A. M. Reina. Separation of Concerns and Model Driven Development Applied to Web Systems. *AOSD'06 Student Extravaganza: Poster Session. Held in conjunction with the Aspect-Oriented Software Development Conference (AOSD'06)*. Bonn, Germany. March, 2006.

2005

[JISBD-DSDM'05] A. M. Reina, J. Torres. Implicaciones de transformaciones oblicuas en e desarrollo de un framework generador de aplicaciones orientadas a aspectos. *II Taller sobre Desarrollo de Software Dirigido por Modelos, MDA y Aplicaciones (DSDM'05) published as CEUR Workshop Proceedings*, Vol. 157. September 2005. ISSN: 1613-0073.

[JISBD-DSOA'05] A. M. Reina, J. Torres. Hacia la separación de enlaces en sistemas web. *Tercer Taller de Desarrollo de Software Orientado a Aspectos (DSOA05). Celebrado junto con las Jornadas de Ingeniería del Software y Bases de Datos (2005), dentro del Primer Congreso Español de Informática (CEDI)*. Granada, España. September, 2005. Publicado por la Universidad de Extremadura como Informe Técnico TR-24/05. pp. 11–17. ISBN: 84-7723-670-4. Eds. Joäo Araújo, Elena Navarro, Mónica Pinto, Fernando Sánchez.

[ECOOP-MAHCC'05] A. M. Reina, J. Torres. Weaving AspectJ by means of transformations. In *Proceedings of the First Workshop on Models and Aspects - Handling Crosscutting Concerns in MDSD. Workshop held at the 19th European Conference on Object-Oriented Programming (ECOOP 2005)*, July 2005.

2004

[JISBD-DSOA'04] A. M. Reina, J. Torres, M. Toro, M. J. Escalona. Modelando aspectos con lenguajes específicos de dominio. In Lidia Fuentes, Ana Moreira, Juan Manuel Murillo, editors, *Taller de Desarrollo de Software Orientado a Aspectos (DSOA04)*, Publicado por la Universidad de Extremadura como Informe Técnico TR-23/04. pp. 91–98. ISBN: 84-688-8889-3. November, 2004.

[UML-AOM'04] A. M. Reina, J. Torres, M. Toro. Separating concerns by means of UML-profiles and metamodels in PIMs. In Omar Aldawud, Grady Booch, Jeff Gray, Jörg Kienzle, Dominik Stein, MohamedKandé, Faisal Akkawi, Tzilla Elrad, editors. *The 5th Aspect-Oriented Modeling Workshop In Conjunction with UML 2004*, October 2004.

[JISBD-DSDM'04] A. M. Reina, J. Torres, M. Toro, J. A. Álvarez. Separación de conceptos y MDA: Arquitectura de un framework. In Vicente Pelechano, Antonio Vallecillo, Javier Muñoz, Joan Fons, editors, *I Taller sobre Desarrollo de Software Dirigido por Modelos, MDA y Aplicaciones (DSDM'04)*. Publicado por la Universidad Politécnica de Valencia. ISBN: 84-688-8870-2. November 2004.

[RCC'04] A. M. Reina, J. Torres. Components+aspects: A general overview. *Revista Colombiana de Computación*, 5(1):77–95, June 2004. ISSN: 1657-2831.

2003

[IADIS-WWW'03a] A. M. Reina, J. Torres, M. J. Escalona, J. A. Ortega. Revisiting requirements in web modelling languages. In P. Isaías, N. Karmakar, editors, *Proceedings of the IADIS International Conference WWW/Internet*, Vol. 2, pages 1065–1067. IADIS Press, November 2003.

[IADIS-WWW'03b] A. M. Reina, J. Torres, M. Toro, J. A. Álvarez. Concerns vs. components for web development. In P. Isaías, N. Karmakar, editors, *Proceedings of the IADIS International Conference WWW/Internet*, Vol. 2, pages 873–876. IADIS Press, November 2003.

[JISBD-DSOA'03] A. M. Reina, J. Torres, M. Toro, J. A. Álvarez, J. M. Nieto. Una experiencia práctica reutilizando aspectos. In Lidia Fuentes, Juan Hernández, Ana Moreira, editors, *Taller sobre Desarrollo de Software Orientado a Aspectos (DSOA03). Taller celebrado en conjunción con las VIII Jornadas de Ingeniería del Software y Bases de Datos.*. Publicado por la Univ. de Extremadura como Informe Técnico TR20/2003, pages 3–10, November 2003.

[JISBD-NTICCSW'03] A. M. Reina, J. Torres. Experiencias Prácticas con Aspectos y Componentes en el Desarrollo Web. *Taller sobre Nuevas Tecnologías de la Información: Componentes y Servicios Web (NTICCSW03). Taller celebrado en conjunción con las VIII Jornadas de Ingeniería del Software y Bases de Datos.* Alicante, España. Noviembre, 2003. Publicado por la Univ. de Extremadura como

Informe Técnico TR21/2003. pages. 11–20. Eds. P.J. Clemente, J. M. Corchado, R. Corchuelo, J. Pavón, D. Sevilla.

[ECOOP-AAOS'03] A. M. Reina, J. Torres, M. Toro. Aspect-oriented web development vs. non aspect-oriented web development. In *AAOS 2003:Analysis of Aspect-Oriented Software (ECOOP 2003)*, July 2003.

[DOLMEN'03a] A. M. Reina, J. Torres, M. Toro, J. A. Álvarez. M. J. Escalona. La Separación de Conceptos en Sistemas Web. *Actas de las IV Jornadas Dolmen.* pages. 137–142. Alicante, España. Noviembre, 2003.

2002

[JISBD-THSCA'02] A. M. Reina, J. Torres, M. Toro, M. J. Escalona. Caracterizando el aspecto de navegación. *Taller de Hipermedia de Sistemas Colaborativos y Adaptativos.* El Escorial, Madrid, España. Noviembre 2002.

[DDMA'02] A. M. Reina, J. Torres. Separating the navigational aspect. In Mehmet Akşit, Ziéd Choukair, editors, *Proc. 2nd Int'l Workshop on Aspect Oriented Programming for Distributed Computing Systems (ICDCS-2002), Vol. 2*, July 2002.

[GAOP'02] A. M. Reina, J. Torres. Analysing the navigational aspect. In Pascal Costanza, Günter Kniesel, Katharina Mehner, Elke Pulvermüller, Andreas Speck, editors, *Second Workshop on Aspect-Oriented Software Development of the German Information Society.* Institut für Informatik III, Universität Bonn, February 2002. Technical report IAI-TR-2002-1, ISSN 0944-8535.

[DOLMEN'02b] A. M. Reina, J. Torres, M. Toro, M. J. Escalona, J. M. Marquez. Desacoplando clases navegacionales de clases conceptuales. *Actas de las III Jornadas Dolmen.* El Escorial, Madrid, España. Noviembre, 2002.

[DOLMEN'02a] A. M. Reina, J. Torres, M. J. Escalona, J. A. Ortega.. La navegación y la separación avanzada de conceptos. *Actas de las II Jornadas Dolmen.* Valencia, España. Junio, 2002.

2000

[LSI-TR'00] A. M. Reina. Visión general de la programación orientada a aspectos. Technical Report LSI-2000-11, Departamento de Lenguajes y Sistemas Informáticos, November 2000.

A.2 Publicaciones relacionadas

2011

[JSS'11] S. Pozo, R. M. Gasca, A. M. Reina, A. J. Varela. CONFIDDENT: A Model-Driven Consistent and Non-Redundant Layer-3 Firewall ACL Design, Development and Maintenance Framework. *Journal of Systems and Software (JSS)*. ISSN: 0164-1212. Aceptada para publicación.

[IJCIS'11] R. Z. Frantz, A. M. Reina, R. Corchuelo. A Domain-Specific Language To Design Enterprise Application Integration Solutions. *International Journal of Cooperative Information Systems (IJCIS)*. 20(2). pages. 143-176. ISSN: 0218-8430.

[ECMFA-MELO'11] M. T. Gómez-López, A. M. Reina, R. M. Gasca. Model-Driven Engineering for Constraint Database Query Evaluation. *Workshop on Model-Driven Engineering, Logic and Optimization: friends or foes? (MELO'11)*. Birmingham, UK.

2006

[JISBD-DSDM'06] J. J. Gutiérrez, M. J. Escalona, M. Mejías, A. M. Reina. Modelos de Pruebas para Pruebas del Sistema. *Actas del Taller sobre Desarrollo de Software Dirigido por Modelos. MDA y Aplicaciones*. Sitges, Spain. October, 2006. CEUR Workshop Proceedings. Vol. 227.

2005

[JSPTIN'05] J. Torres, M. J. Escalona, A. M. Reina. TIC2003-00369. Navegación e Interración con el Usuario en el Desarrollo de Sistemas de Información Web: Métodos, Técnicas y Herramientas. *Actas de las Jornadas de Seguimiento de Proyectos en Tecnologías Informáticas 2005 (jspTIN2005)*. Granada, España. Septiembre, 2005.

2004

[OOSPLA-EA'04] M. J. Escalona, A. M. Reina, J. Torres, M. Mejías. The Navigational Aspect in the Requirement Specification of NDT. *Proceedings of the Early Aspects 2004: Aspect-Oriented Requirements Engineering and Architecture Design Workshop*. Vancouver, Canada. October, 2004.

[IADIS-WWW'04] M. J. Escalona, A. M. Reina, J. Torres, M. Mejías. The Navigational Aspect in the Requirement Specification of NDT. *Proceedings of the IADIS International Conference WWW/Internet 2004*. pages. 1237-1238. ISBN: 972-99353-0-0. Madrid, Spain. October, 2004.

2003

[DOLMEN'03b] J. A. Ortega, J. M. Márquez, A. M. Reina, J. A. Álvarez. Definición y Representación de Reglas de Transformación de un Modelo Independiente en otro Específico de la Plataforma. *Actas de las IV Jornadas Dolmen*. pages 124-130. Alicante, España. Noviembre, 2003.

[DOLMEN'03c] J. Torres, M. J. Escalona, A. M. Reina, M. Mejías, J. A. Ortega, J. M. Cordero, M. González, J. A. Álvarez. Informe Final Subproyecto MADEIRA. Metodologías Avanzadas para el Desarrollo en Sistemas Web. Proyecto DOLMEN. *Actas de las IV Jornadas Dolmen.* pages 46-52. Alicante, España. Noviembre, 2003.

[ICWE'03a] M. J. Escalona, M. Mejías, J. Torres, A. M. Reina. NDT-Tool: A CASE Tool to deal with Requirements in Web Information Systems. *Proceedings of the International Conference on Web Engineering (ICWE 2003).* Oviedo, España. July, 2003. Lecture Notes in Computer Science (LNCS 2722). ISSN: 0302-9743. pages 212-213. Ed. Springer-Verlag.

[ICWE'03b] M. J. Escalona, M. Mejías, J. Torres, A. M. Reina. The NDT Development Process. *Proceedings of the International Conference on Web Engineering (ICWE 2003).* Oviedo, España. July, 2003. Lecture Notes in Computer Science (LNCS 2722). ISSN: 0302-9743. pages 463-467. Ed. Springer-Verlag.

2002

[JISBD-TISOW'02] M. J. Escalona, M. Mejías, J. Torres, A. M. Reina. Desarrollo de la navegación en entornos web. *Actas del II Taller en Ingeniería del Software Orientada al Web.* El Escorial, Madrid, España. Noviembre, 2002.

[DOLMEN'02d] M. J. Escalona, M. Mejías, J. Torres, A. M. Reina. Resultado comparativo de las actuales metodologías para la web. *Actas de las III Jornadas Dolmen.* El Escorial, Madrid, España. Noviembre, 2002.

[IDEAS'02] M. J. Escalona, M. Mejías, J. Torres, A. M. Reina. NDT - una técnica para el desarrollo de la navegación. *Actas del 5º Workshop Iberoamericano de Intenieria de Requisitos y Ambientes Software.* La Habana, Cuba. Abril, 2002.

[DOLMEN'02c] M. J. Escalona, M. Mejías, J. Torres, A. M. Reina. Definición de requisitos de interacción. *Actas de las II Jornadas Dolmen.* Valencia, España. Junio, 2002.

2001

[DOLMEN'01] M. J. Escalona, M. Mejías, J. Torres, A. M. Reina. Propuesta metodológica para el desarrollo de sistemas para el tratamiento de bibliotecas digitales. *Actas de las I Jornadas Dolmen.* Sevilla, España. Junio, 2001.

[JBIDI'01] M. J. Escalona, M. Mejías, J. Torres, A. M. Reina, J. M. Cordero Requisitos de almacenamiento de información identificación de actores para una biblioteca digital de bienes muebles. *Actas de las II Jornadas de Bibliotecas Digitales.* Almagro, Ciudad Real, España. Noviembre, 2001.

B

Listado detallado de citas

En este apéndice se muestra un listado detallado de aquéllas publicaciones que han referenciado a algún trabajo nuestro. Las publicaciones que se han citado se han ordenado por fecha de publicación, empezando por las más recientes. Dentro de cada publicación, las citas se han agrupado por tipo, diferenciando así aquellas publicaciones realizadas en revistas, de las realizadas en congresos y talleres, las de libros, los informes técnicos, las tesis, los proyectos fin de carrera y otros.

[AEWSE'07] Improving the Adaptation of Web Applications to Different Versions of Software with MDA [335] (1 cit.)

- Citas en Congresos y Talleres

  1. J. C. Castrejón, R. Lopez-Landa, R. Lozano. Model2Roo: A model driven approach for web application development based on the Eclipse Modeling Framework and Spring Roo. In *Proceedings of 2011 21st International Conference on Electrical Communications and Computers (CONIELECOMP)* . IEEE, Mar 2011.

[ENTCS'07] Using aspect-oriented techniques to improve the reuse of metamodels [327] (7 cit.)

- Citas en Revistas

  1. I. García-Magariño, R. Fuentes-Fernández, and J. J. Gómez-Sanz. Guideline for the definition of EMF metamodels using an entity-relationship approach. *Information and Software Technology*, 51(8):1217–1230, Aug 2009.

- Citas en Congresos y Talleres

1. S. Haschemi and A. Wider. An extensible language for service dependency management. In *Proceedings of the 35th EUROMICRO Conference on Software Engineering and Advanced Applications, 2009 (SEAA'09)*. IEEE, Aug 2009.
2. J. Zhang, Y. Chen, H. Li, and G. Liu. Research on aspect-oriented modeling in the framework of MDA. In *Proceedings of the 2nd IEEE International Conference on Computer Science and Information Technology (ICCSIT'09)*, pages 108–111. IEEE, Aug 2009.
3. J. Zhang, Y. Chen, G. Liu and H. Li. Using Sequence Diagram to Support Aspect-Oriented Programming in MDA. In *International Conference on Intelligent Huma-Machine Systems and Cybernetics, 2009 (IHMSC'09)*, Vol. 1, pages 359–362. IEEE, 2009. ISBN: 978-0-7695-3752-8
4. A. Bergmayr Reuse in Modelling Method Development based on Metamodelling. In *Proceedings of the MODELS 2010 Doctoral Symposium.*, pages 7–12. 2010
5. A. Bergmayr ReuseMe - Towards Aspect-Driven Reuse in Modelling Method Development. In *MODELS 2010 Workshops.*, LNCS 6627, pages. 4- 18. 2011

- Citas en Tesis

1. I. García-Magariño García. Un Marco para la Definición y Transformación de Modelos en los Sistemas Multiagentes. Phd. Thesis. Departamento de Ingeniería del Software e Inteligencia Artificial Facultad de Informática. Universidad Complutense de Madrid 2010

[JISBD-DSOA'06] Hacia Lenguajes de Metamodelado Orientados a Aspectos [333] (1 cit.)

- Citas en Informes Técnicos

1. H. López, F. Varesi, M. Viñolo, D. Calegari, C. Luna. Estado del Arte de Lenguajes y Herramientas de Transformación de Modelos. Reporte Técnico RT 09-19. PEDECIBA. Instituto de Computación. Facultad de Ingeniería Universidad de la República 2009 ISSN 0797-6410

[ECOOP-MAHCC'05] Weaving AspectJ Aspects by means of Transformations [326] (7 cit.)

- Citas en Revistas

1. M. Pitanga Alves, P. F. Pires, F. C. Delicato, and M. L. M. Campos. CrossMDA: A Model-Driven Approach for Aspect Management. *Journal of Universal Computer Science*, 14(8):1314–1343, 2008.

- Citas en Congresos y Talleres

1. M. Pitanga Alves, P. F. Pires, and M. L. M. Campos F. C. Delicato. Crossmda: Arcabouço para integração de interesses transversais no desenvolvimento orientado a modelos. In *Proceedings of the Simpósio Brasileiro de Componentes, Arquiteturas e Reutilizaçao de Software(SBCARS 2007)*, pages 177–190, Aug 2008.

- Citas en Tesis

1. T. Vanderka. Prekonanie rozdielov medzi aspektovo orientovanými jazykmi pomocou prístupu MDA (Overcomming the differences between aspect oriented languages with MDA). Diploma Thesis. Slovenská technická univerzita v Bratislave. Fakulta Informatiky a Informacných Technológií. May, 2007
2. M. Pitanga Alves. CrossMDA: Arcabouço para integração de interesses transversais no desenvolvimento orientado a modelos. Dissertação de Mestrado. Universidade Federal do Rio de Janeiro. Instituto de Matemática. Núcleo de Computação Eletrônica. 2007
3. A. I. Schmied. Program Transformations and their Semi-Automatic Composition A Pragmatic Approach to Invasive Middleware Application Engineering. Phd. Thesis. Fakultät für Ingenieurwissenschaften und Informatik. Institut für Verteilte Systeme Universität Ulm. 2009
4. G. C. de Araújo Filgueira. CrossMDA-SPL: Uma Abordagem para Gerêrencia de Variabilidades Dirigida por Modelos e Aspectos. Dissertaçao de Mestrado. Universidad do Rio Grande do Norte. Centro de Ciências Exatas e da Terra. Departamento de Informática e Matemática Aplicada. 2009

- Otros

1. G. C. Torres Díaz. UML: Panacea or Malady? In *MSDN Library*. Sept,2008. url:`http://msdn.microsoft.com/en-us/library/cc948344.aspx`

[RCC'04] Components+Aspects: A General Overview [324] (3 cit.)

- Citas en Revistas

1. M. Dragone, H. Jordan, D. Lillis, R. W. Collier. Separation of concerns in hybrid component and agent systems. *International Journal of Communication Networks and Distributed Systems*, 6(2):176–201, 2011.

- Citas en Tesis

1. R. Gupta. A wrapper api for reflex. Thesis for the degree of master of science in computer science, Vrije Universiteit Brussel. Belgium Faculty of Sciences, 2005.
2. Juan J. Bastidas B. Desarrollo de un sistema de componentes de software utilizando inteligencia artificial colectiva. Tesis de Grado para Ingeniero de Sistemas. Dpto. de Computación. Facultad de Ingeniería. Escuela de Sistemas. Universidad de Los Andes. Mérida. Estado de Mérida. 2006

[UML-AOM'04] Developing Generic Solutions with Aspects [332] (25 cit.)

- Citas en Revistas

1. A. Solberg, D. Simmonds, R. Reddy, R. France, and S. Ghosh. Developing distributed services using an aspect oriented model driven framework. *International Journal of Cooperative Information Systems*, 15(4):535–564, 2006.
2. P. Amaya, C. González, and J. M. Murillo. Towards a subject-oriented model-driven framework. Electronic Notes in Theoretical Computer Science 163(1):31–44, 2006.
3. L. Nemuraite and M. Balandyte. Aspect-Oriented Use Cases and Crosscutting Interfaces for Reconfigurable Behavior Modeling. *Frontiers in Artificial Intelligence and Applications*, 15:189–202, 2007.

- Citas en Congresos y Talleres

1. P. Amaya, C. González, and J. M. Murillo. MDA and separation of aspects - an approach based on multiple views and subject oriented design. In *Proceedings of the 6th International Workshop on Aspect-Oriented Modeling, held in conjunction with the 4th International Conference on Aspect-Oriented Software Development (AOSD'05)*, March 2005.
2. J. Conejero, J. Hernández, and R. Rodríguez. UML profile definition for dealing with the notification aspect in distributed environments. In *Proceedings of the 6th International Workshop on Aspect-Oriented Modeling, held in conjunction with the 4th International Conference on Aspect-Oriented Software Development (AOSD'05)*, March 2005.
3. P. Amaya, C. González, and J. M. Murillo. AspectMDA: Hacia un desarrollo incremental consistente integrando MDA y orientación a aspectos. En *Actas del II Taller sobre Desarrollo de Software Dirigido por Modelos, MDA y Aplicaciones (DSDM'05), celebrado en el marco de las X Jornadas de Ingeniería del Software y Bases de Datos*. Publicado por CEUR Workshop Proceedings. Vol. 157 ISSN: 1613-0073. Granada, España. Sep, 2005

4. D. Simmonds, A. Solberg, R. Reddy, R. France, and S. Ghosh. An aspect oriented model driven framework. In *Proceedings of the Ninth IEEE The Enterprise Computing Conference (EDOC 2005)*. IEEE, Sept 2005.

5. A. Solberg, D. Simmonds, R. Reddy, S. Ghosh, and R. France. Using aspect oriented technologies to support separation of concerns in model driven development. In *Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC 2005)*, July 2005.

6. A. Schauerhuber, W. Schwinger, W. Retschitzegger, and M. Wimmer. Towards a common reference architecture for aspect-oriented modeling. In *Proceedings of the 8th International Workshop on Aspect-Oriented Modeling (AOSD'06), held in conjunction with the 5th International Conference on Aspect-Oriented Software Development (AOSD'06)*, March 2006.

7. M. Balandyte, L. Nemuraite Aspect-oriented Modelling in Information Systems Development. In *Informacines technologijos 2006 (IT 2006)*, pp. 623-631. 2006

8. M. Balandyte and L. Nemuraite. Time, event and self-management aspects in model-driven development of information systems. In *Proceedings of the 7th International Baltic Conference on Databases and Information Systems 2006*, pages 151–158. IEEE Press, 2006.

9. J. Evermann. A meta-level specification and profile for aspectj in UML. In *AOM '07: Proceedings of the 10th international workshop on Aspect-oriented modeling*, pages 21–27, New York, NY, USA, 2007. ACM Press.

10. A. Przybylek. Separation of crosscutting concerns at the design level: an extension to the UML metamodel. In M. Ganzha, M. Paprzycki, and T. Pelech-Pilichowski, editors, *Proceedings of the International Multiconference on Computer Science and Information Technology*, volume 3, pages 551–557. IEEE, 2008.

11. F. E. Alam, J. Evermann and A. Fiech Modeling for Dynamic Aspect-Oriented Development. *ACM International Conference Proceeding Series. Proceedings of the 2009 C3S2E Conference* Montreal, Quebec, Canada POSTER SESSION: Student posters and demos pp. 143-147. 2009. ISBN: 978-1-60558-401-0

12. J. Zhang, Y. Chen, H. Li, and G. Liu. Research on aspect-oriented modeling in the framework of MDA. In *Proceedings of the 2nd IEEE International Conference on Computer Science and Information Technology (ICCSIT'09)*, pp. 108–111. IEEE, Aug 2009.

13. D. Mouheb, C. Talhiand, A. Mourad, V. Lima, M. Debbabi, L. Wang, M. Pourzandi. An Aspect-Oriented Approach for Software Security Hardening: from Design to Implementation. In *Proceeding of the 2009 conference on New*

Trends in Software Methodologies, Tools and Techniques: Proceedings of the Eighth SoMeT'09, pp. 203–222. IOS Press, 009. ISBN:978-1-60750-049-0.

14. A. Solberg. Manage QoS in MDD using an aspect-oriented model driven framework. In *Interoperability for Enterprise Software and Applications: Proceedings of the Workshops and the Doctorial Symposium of the Second IFAC/IFIP I-ESA International Conference: EI2N, WSI, IS-TSPQ 2006*. Eds H. Panetto and N. Boudjlida. pp. 255-264 ISTE, London, UK 2010
15. Z. Sharafi, P. Mirshams, A. Hamou-Lhadj and C. Constantinides. Extending the UML Metamodel to Provide Support for Crosscutting Concerns. In *Proceedings of the ACIS International Conference on Software Engineering Research, Management and Applications*, pages 149–157. IEEE, May 2010.
16. J. Evermann, A. Fiech, F. E. Alam. A Platform-Independent UML Profile for Aspect-Oriented Development. In *Proceedings of The Fourth International C* Conference on Computer Science and Software Engineering (C3S2E '11)*, pages 25–34. ACM, 2011.
17. R. M. Garg and D. Dahiya An Aspect Oriented Component Based Model Driven Development In *Proceedings of the Seconcd International Conference on Software Engineering and Computer Systems (ICSECS 2011)*, pages 502–517. Communications in Computer and Information Science. Volume 181, Part 5. Springer Verlag, 2011.

- Citas en Informes Técnicos

1. R. Chitchyan, A. Rashid, P. Sawyer, A . Garcia, M. Pinto Alarcón, J. Bakker, B. Tekinerdogan, and A. Jackson and S. Clarke. Survey of aspect-oriented analysis and design approaches. Technical Report AOSD-Europe-ULANC-9, AOSD-Europe, May 2005.
2. A. Schauerhuber, W. Schwinger, E. Kapsammer, W. Retschitzegger, M. Wimmer, and G. Kappel. A survey on aspect-oriented modeling approaches. Technical Report Technical Report, Vienna University of Technology, April 2007.

- Citas en Tesis

1. A. Schauerhuber. *Applying Aspect-Orientation to the Model-Driven Development of Ubiquitous Web Applications*. PhD thesis, Faculty of Informatics. Vienna University of Technology, Nov 2007.
2. W. Schult. *Architektur komponentenbasierter Systeme mit LOOM: Aspekte, Muster, Werkzeuge* Dissertation zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften. Hasso-Plattner-Institut für Softwaresystemtechnik. Universität Potsdam, Jun 2009.

3. P. M. Shahshahani. *Extending the Knowledge Discovery Metamodel to Support Aspect-Oriented Programming* Dissertation For the Degree of Master of Applied Science in Software Engineering Concordia University. Department of Computer Science and Software Engineering. Montreal, Quebec, Canada, Apr., 2011.

[ECOOP-AAOS'03] Aspect-Oriented Web Development vs. Non Aspect-Oriented Web Development [331] (12 cit.)

- Citas en Revistas

1. A. Marchetto. A Concerns-Based Metrics Suite for Web Applications. *INFOCOMP - Journal of Computer Science*, 4(3):11–22, Sept 2005.
2. C. Bellettini, A. Marchetto, and A. Trentini. An approach to concerns and aspects mining for web applications. *International Journal Of Information Technology*, 2(1):12–20, 2005.
3. C. Bellettini, A. Marchetto, and A. Trentini. Multi-dimensional concerns mining for web applications via concept-analysis. *Transactions on Engineering, Computing and Technology*, 4:129–132, February 2005. ISSN: 1305-5313.

- Citas en Congresos y Talleres

1. F. Tessier, M. Badri, and L. Badri. Développement orienté aspect: Survol de l'État de lárt, Évaluation et perspectives. In *Proceedings of ACFAS 2004*, 2004.
2. G. Fiumara, M. La Rosa, T. Pimpo. (X)querying RSS/Atom Feeds Extracted from News Web Sites: a Cocoon-based Portal. In *Proceedings of the Workshop BOF - Between Oontologies and Folksonomies: Tools and Architectures for Managing and Retrieving Emerging Knowledge in Communities (BOF'07)*, CEUR Workshop Proceedings, Vol. 312, 10–20. June, 2007. Editors, D. Maggiorini, A. Provetti, L. Ripamonti
3. J. Esparteiro Garcia. Aspect-oriented web development in PHP. In *Proceedings of the DSIE08 - Doctoral Symposium on Informatics Engineering 2008*, Feb 2008.
4. D. Altunbay, D. Arifoglu, H. Zitouni. Aspect-Oriented Development of an E-Commerce Application. In *Proceedings of the Third Turkish Aspect-Oriented Software Development Workshop (TAOSD2008)*, 53–64. Ed. B. Tekinerdogan. Dec, 2008.
5. D. Sarikaya, C. Ufuk Hantas, D. Demirbas. Comparative Analysis Of Strategies For Applying AOP On Legacy Code. In *Proceedings of the Fourth Turkish Aspect-Oriented Software Development Workshop (TAOSD2009)*, 68–78. Ed. B. Tekinerdogan. Dec, 2009.

- Citas en Informes Técnicos

  1. M. Han and C. Hofmeister. Separating and representing navigation concerns in web applications. Technical Report 04-004, Lehigh University, 2004.
  2. M. Han and C. Hofmeister. Modelling navigation routing in J2EE web applications. Technical Report 04-011, Lehigh University, 2004.

- Citas en Tesis

  1. F. Tessier. Assurance qualité et développement de logiciels orientés aspect : Détection de conflits entre les aspects basée sur les modèles. Thesis de maîtrise, Université du Québec à Trois-Rivières, Jun 2005.
  2. André Janus. Konzepte der aspekt-orientierten programmierung im komponentenbasierten web engineering. Diploma thesis, Fakultät für Informatik. Universität Karlsruhe (TH), 2006.

[JISBD-DSOA'03] Una Experiencia Práctica Reutilizando Aspectos [340] (3 cit.)

- Citas en Revistas

  1. I. Castillo, F. Losavio, A. Matteo and J. Boegh. REquirements, Aspects and Software Quality: the REASQ model. In *Journal of Object Technology (JOT)*. 9(4):69–91. Jul, 2010. ISSN: 16601769.

- Citas en Informes Técnicos

  1. J. Chuico and W. Vásquez. Definición e implementación de los lineamientos para el paradigma de programación orientada a aspectos en el desarrollo de software. Technical report, Universidad Técnica Particular de Loja, 2007.

- Citas en Tesis

  1. J. Chuico and W. Vásquez. Definición de lineamientos para la Programación Orientada a Aspectos. Tesis de Grado. Universidad Técnica Particular de Loja, 2007

[DDMA'02] Separating the Navigational Aspect [323] (6 cit.)

- Citas en Revistas

  1. R. Rodríguez, F. Sánchez, J. M. Conejero, and J. Pedrero. Modelando procesos de negocio web desde una perspectiva orientada a aspectos. *IEEE América Latina*, 3(1), Mar 2005.

2. R. Rodríguez, F. Sánchez, J. M. Conejero, and J. Pedrero. Modelando procesos de negocio web desde una perspectiva orientada a aspectos. *Ciencia y Técnica Administrativa ejournal*, 4(22), May/Jun 2005.
3. A. Marchetto. A Concerns-Based Metrics Suite for Web Applications. *INFOCOMP - Journal of Computer Science*, 4(3):11–22, Sept 2005.

- Citas en Congresos y Talleres

1. J. Esparteiro Garcia. Aspect-oriented web development in PHP. In *Proceedings of the DSIE08 - Doctoral Symposium on Informatics Engineering 2008*, Feb 2008.

- Citas en Tesis

1. M. B. Hankerson. Towards a taxonomy of aspect-oriented programming. Thesis for the degree master of science in computer science, Department of Computer Science and Information Sciences East Tennessee State University, Dec 2003.

[GAOP'02] Analysing the Navigational Aspect [322] (6 cit.)

- Citas en Revistas

1. L. Xudong, J. Hsien, and Lulei. Web navigation in the application of the model review. *Journal of Computer Applications*, 26(1):244–247, 2006.

- Citas en Congresos y Talleres

1. M. Han and C. Baufmeister. Separation of navigation routing code in j2ee web applications. In *Proceedings of the 5th International Conference on Web Engineering (ICWE 2005)*, volume 3579, pages 406–416. Springer-Verlag Lecture Notes in Computer Science, 2005.
2. F. Lyardet, G. Rossi, and S. Gordillo. Views, aspects and roles in web applications design. In *VAR 05. First Workshop on Views, Aspects and Roles, in ECOOP 2005*, New York, NY, USA, July 2005.

- Citas en Informes Técnicos

1. M. Han and C. Hofmeister. Separating and representing navigation concerns in web applications. Technical Report 04-004, Lehigh University, 2004.
2. M. Han and C. Hofmeister. Modelling navigation routing in J2EE web applications. Technical Report 04-011, Lehigh University, 2004.

- Otros

1. Downloadable Reference Library. Chapter 18. Analysing modelling for web applications. Web:`http://www.rspa.com/reflib/AnalysisWebApps.html` R.S. Pressman & Associates, Inc.

[LSI-TR'00] Visión General de la Programación Orientada a Aspectos [317] (17 cit.)

- Citas en Revistas

1. M. F. Rosique Contreras. Pada: Patrón de distribución con aspectos. *Antena de Telecomunicación*, (169), Sept 2007.
2. G. Samaniego and J. Pulido. Composición de requerimientos no funcionales. *Paradigma . Revista electrónica en Construcción de Software*, 2(2), 2008.

- Citas en Congresos y Talleres

1. N. Cova Suazo and J.O.O. Aguirre. Aspect-oriented web services orchestration. In *Proceedings of the 2nd International Conference on Electrical and Electronics Engineering 2005*. IEEE Press, Sept 2005.
2. L. S. Baigorria and G. Montejano. Métricas aplicadas a la programación orientada a aspectos. In *Workshop de Investigadores en Ciencias de la Computación WICC-2006*, Jun 2006.
3. N. Debnath, L. Baigorria, D. Riesco, and G. Montejano. Metrics applied to aspect oriented design using uml profiles. In *Proceedings of the IEEE Symposium on Computers and Communications (ISCC 2008).*, pages 654–657, 2008.

- Citas en Libros

1. F. Gutiérrez, F. Durán, and E. Pimentel. *Programación Orientada a Objetos con Java*, capítulo 1. Ed. Thomson Paraninfo, 2007.

- Citas en Informes Técnicos

1. R. Almonacid Arismendi, S. Hernández Venega. Programación Orientada a Aspectos. Informe final de habilitación profesional. Universidad del Bío-Bío. Facultad de Ciencias Empresariales. Departamento de Sistemas de Información. Chile: Universidad del Bío Bío, 2008.

- Citas en Tesis

1. F. Asteasuain and B. E. Contreras. Programación orientada a aspectos. Análisis del paradigma. Tesis de licenciatura, Departamento de Ciencias e Ingeniería de la Computación. Universidad Nacional del Sur, Oct 2002.

2. E. M. Alférez Salinas and G. H. Alférez Salinas. Análisis de un sistema de información bajo la aproximación de la orientación a aspectos. caso práctico: Manejo de solicitudes en la mesa de ayuda de servicios generales de una universidad. Tesis de grado, Departamento Informática y Sistemas. Universidad Eafit, Oct 2004.
3. P. A. Burgos Heredia and J. A. Pinzón Ortíz. API de comunicaciones para pocketpc basado en la programación orientada a aspectos. Tesis de grado, Pontificia Universidad Javeriana. Facultad de Ingeniería, Jun 2005.
4. N. N. Cova Suazo. Orquestación de servicios web orientada a aspectos. Tesis de maestría, Departamento de Ingeniería Eléctrica. Sección de Computación. Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional Mexico, D.F., Oct 2005.
5. L. Vinuesa Martínez. *Separación Dinámica de Aspectos Independiente del Lenguaje y Plataforma Mediante el Uso de Reflexión Computacional.* Tesis doctoral, Universidad de Oviedo, Oct 2007.
6. C. Córdova. *Implementación de requisitos no funcionales a través de la programación orientada a aspectos.* Tesis (Ingeniero de Sistemas). Universidad Nacional Abierta, Centro Local Sucre, 2007
7. G. H. Alférez Salinas. *Aspect-Oriented Driven Variability in Software Product Lines.* Thesis for the Degree Master of Science in Information and Communication Technology, Assumption University of Thailand, Jan 2008.
8. P. A. Marcial Palafox. *Modelado de tácticas de atributos de calidad para la generación de arquitecturas ejecutables.* Tesis de Maestría, Universidad Autónoma Metropolitana, Iztapalapa, Mexico, 2009.

- Citas en Proyecto Fin de Carrera

1. M. F. Rosique Contreras. La programación distribuida de aplicaciones desde la perspectiva de la programación orientada a aspectos: Estudio y demostración de uso. Proyecto fin de carrera, Escuela Técnica Superior de Ingeniería de Telecomunicación. Universidad Politécnica de Cartagena, Dec 2003.
2. S. Manzanares Guillén. Programación orientada a aspectos.una experiencia práctica con AspectJ. Proyecto fin de carrera, Departamento de Informática y Sistemas. Facultad de Informática de la Universidad de Murcia, Jun 2005.

Bibliografía

[1] *DSAL'06: Proceedings of 1st Workshop on Domain Specific Aspect Languages*, 2006.

[2] *DSAL'07: Proceedings of 2nd workshop on Domain Specific Aspect Languages*, New York, NY, USA, 2007. ACM Press.

[3] A. Abouzahra, J. Bézivin, M. D. Fabro, F. Jouault. A Practical Approach to Bridging Domain Specific Languages with UML Profiles. En *Proceedings of the 10th International Conference on Best Practices for Model Driven Software Development*, 2005.

[4] M. Akşit, editor. *Proceedings ot 2nd International Conference on Aspect-Oriented Software Development (AOSD-2003)*. ACM Press, marzo 2003.

[5] M. Akşit, J. Bosch, W. van der Sterren, L. Bergmans. Real-Time Specification Inheritance Anomalies and Real-Time Filters. En *Proceedings of 8th European Conference on Object-Oriented Programming*, páginas 386–407. Springer Verlag LNCS 821, julio 1994.

[6] M. Alalfi, J. R. Cordy, T. R. Dean. A Survey of Analysis Models and Methods in Website Verification and Testing. Informe técnico 2007-532, School of Computing. Queen's University. Kingston, Ontario, Canada, febrero 2007.

[7] M. Alanen, I. Porres. Coral: A Metamodel Kernel for Transformation Engines. En *Proceedings of the Second European Workshop on Model Driven Architecture (MDA) (EWMDA-2)*, páginas 165–171, setiembre 2004.

[8] O. Aldawud, A. Bader, T. Elrad. Weaving with Statecharts. En AOSD-UML'02 [17].

[9] O. Aldawud, G. Booch, J. Gray, J. Kienzle, D. Stein, M. Kandé, F. Akkawi, T. Elrad, editores. *5th Aspect-Oriented Modeling Workshop In Conjunction with UML 2004*, octubre 2004.

[10] O. Aldawud, M. Kandé, G. Booch, B. Harrison, D. Stein, editores. *Third International Workshop on Aspect Oriented Modeling*, marzo 2003.

[11] O. Aldawud, M. Kandé, G. Booch, B. Harrison, D. Stein, J. Gray, S. Clarke, A. Z. Santeon, P. Tarr, F. Akkawi, editores. *The 4th AOSD Modeling With UML Workshop*, octubre 2003.

[12] C. Amelunxen, A. Königs, T. Rötschke, A. Schürr. MOFLON: A Standard-Compliant Metamodeling Framework with Graph Transformations. En *Proceedings of the Second European Conference on Model Driven Architecture - Foundations and Applications*, páginas 361–375. Springer-Verlag Lecture Notes in Computer Science, 2006.

[13] M. Amor, A. Garcia, L. Fuentes. AGOL: An Aspect-Oriented Domain-Specific Language for MAS. En *Proceedings of the Early Aspects at ICSE: Workshops in Aspect-Oriented Requirements Engineering and Architecture Design*, EARLYASPECTS'07, páginas 4–11, Washington, DC, USA, 2007. IEEE Computer Society.

[14] E. P. Andersen, T. Reenskaug. System Design by Composing Structures of Interacting Objects. En *Proceedings of ECOOP'92 European Conference on Object-Oriented Programming*, páginas 133–152. Springer-Verlag Lecture Notes in Computer Science, julio 1992.

[15] AndroMDA. AndroMDA JSF Cartridge Profile. `http://www.andromda.org/docs/andromda-cartridges/andromda-jsf-cartridge/profile.html#FrontEndNavigation`, octubre 2010.

[16] AOM Workshop Committee. Aspect-Oriented Modelling. `http://www.aspect-modeling.org`.

[17] *Workshop on Aspect-Oriented Modeling with UML (AOSD 2002)*, marzo 2002.

[18] Aosd.net Community. AOSD Glossary. `http://www.aosd.net/wiki/index.php?title=Glossary`.

[19] Apache. Apache Jakarta Velocity. `http://jakarta.apache.org/velocity/`.

[20] T. Apiwattanapong, A. Orso, M. Harrold. Efficient and Precise Dynamic Impact Analysis Using Execute-After Sequences. En *Proceedings of 27th IEEE and ACM SIGSOFT International Conference on Software Engineering (ICSE 2005)*, 2005.

[21] C. Atkinson, T. Kuhne. Model-Driven Development: A Metamodeling Foundation. *IEEE Software*, 20(5):36–41, 2003.

[22] F. Azam, Z. Li, R. Ahmad. Introducing UML Profile for Modelling Information Architecture of Web Applications. En *Proceedings of the 11th Joint International Computer Conference - JICC 2005*, páginas 245–250, febrero 2005.

[23] O. Barais, J. Klein, B. Baudry, A. Jackson, S. Clarke. Composing Multi-view Aspect Models. En *Composition-Based Software Systems, 2008. ICCBSS 2008. Seventh International Conference on*, páginas 43 –52, febrero 2008.

[24] L. Baresi, S. Colazzo, L. Mainetti, S. Morasca. *Web Engineering*, capítulo 11. W2000: A Modeling Notation for Complex Web Applications, páginas 335–408. Springer, 2006.

[25] L. Baresi, F. Garzotto, M. Maritati. W2000 as a MOF Metamodel. En *Proceedings of 6th World Multiconference on Systemics, Cybernetics and Informatics - Web Engineering track*, julio 2002.

[26] L. Baresi, F. Garzotto, P. Paolini. Extending UML for Modeling Web Applications. En *Proceedings of 34th Hawaii International Conference on Systems Sciences (HICSS 2001)*, enero 2001.

[27] J. P. Barros, L. Gomes. Activities as Behaviour Aspects. En M. Kandé y otros [207].

[28] M. Basch, A. Sánchez. Incorporating Aspects into the UML. En O. Aldawud y otros [10].

[29] P. G. Bassett. *Framing Software Reuse: Lessons from the Real World*. Prentice-Hall, Inc., 1996.

[30] B. Baudry, F. Fleurey, R. France, R. Reddy. Exploring the Relationship between Model Composition and Model Transformation. En *Proceedings of Aspect Oriented Modeling Workshop, in conjunction with MoDELS'05*, octubre 2005.

[31] H. Baumeister, A. Knapp, N. Koch, G. Zhang. Modelling Adaptivity with Aspects. En *Proceedings of 5th International Conference on Web Engineering (ICWE 2005)*, volumen 3579, páginas 406–416. Springer-Verlag Lecture Notes in Computer Science, 2005.

[32] H. Baumeister, N. Koch, L. Mandel. Towards a UML Extension for Hypermedia Design. En *Proceedings of the Unified Modeling Language Conference: Beyond the Standard (UML'99)*, número 1723, páginas 614–629. Springer-Verlag Lecture Notes in Computer Science, 1999.

[33] S. A. Becker, A. Berkemeyer. Rapid Application Design and Testing of Web Usability. *IEEE MultiMedia*, 9:38–46, octubre 2002.

[34] D. Beckett. Turtle: Terse RDF Triple Language. Informe técnico, 2004.

[35] C. Bellettini, A. Marchetto, A. Trentini. An Approach to Concerns and Aspects Mining for Web Applications. *International Journal Of Information Technology*, 2(1):12–20, 2005.

[36] L. Bergmans, M. Akşit. Composing Crosscutting Concerns Using Composition Filters. *Communications ACM*, 44(10):51–57, octubre 2001.

[37] J. Bézivin, C. Brunette, R. Chevrel, F. Jouault, I. Kurtev. Bridging the Generic Modeling Environment (GME) and the Eclipse Modeling Framework (EMF). En *OOPSLA Workshop on Best Practices for Model Driven Software Development*, 2005.

[38] J. Bézivin, G. Hillairet, F. Jouault, I. Kurtev, W. Piers. Bridging the MS/DSL Tools and the Eclipse Modeling Framework. En *Proceedings of the International Workshop on Software Factories at OOPSLA 2005*, 2005.

[39] P. S. Bhogill. An Introduction to Java Server Faces. *Java News Brief*, agosto 2003.

[40] L. Bichler. A Flexible Code Generator in MOF-based Modeling Languages. En *Proceedings of the Second OOSPLA Workshop on Generative Techniques in the Context of Model Driven Architecture*, 2003.

[41] G. S. Blair, L. Blair, A. Rashid, A. Moreira, J. Araújo, R. Chitchyan. Engineering Aspect-Oriented Systems. En R. E. Filman y otros [119], páginas 379–406.

[42] P. Boocock. Jamda Model Compiler Framework. `http://jamda.sourceforge.net/docs/index.html`.

[43] A. Boronat, J. Ángel. Carsí, I. Ramos, P. Letelier. Formal Model Merging Applied to Class Diagram Integration. *ENTCS*, 166:5–26, 2007.

[44] M. Bravenboer, K. T. Kalleberg, R. Vermaas, E. Visser. Stratego/XT 0.16: Components for Transformation Systems. En *Proceedings of 2006 ACM SIGPLAN Workshop on Partial Evaluation and Semantics-based Program Manipulation, 2006, Charleston, South Carolina, USA, 2006*, páginas 95–99. ACM, 2006.

[45] J. Brichau, K. Mens, K. De Volder. Building Composable Aspect-Specific Languages with Logic Metaprogramming. En *1st Conference on Generative Programming and Component Engineering*, volumen 2487 de *lncs*, páginas 110–127, Berlin, 2002. Springer-Verlag.

[46] J. Broekstra. *SeRQL: A second-generation RDF query language*. PhD thesis, Vrije Universiteit, Amsterdam, 2005.

[47] A. W. Brown, J. Conallen, D. Tropeano. *Model-Driven Software Development*, capítulo 1. Introduction: Models, Modeling and Model-Driven Architecture (MDA), páginas 1–17. Springer, 2005.

[48] D. M. Brown. *Communicating Design: Developing Web Site Documentation for Design and Planning*, capítulo 8. Site Maps. New Riders, agosto 2006.

[49] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, T. J. Grose. *Eclipse Modelling Framework: A Developer's Guide*. Addison-Wesley, 2003.

[50] J. Bézivin. On the Unification Power of Models. *Software and System Modeling*, 4(2):171–188, 2005.

[51] J. Bézivin, S. Bouzitouna, M. Del Fabro, M.-P. Gervais, F. Jouault, D. Kolovos, I. Kurtev, R. Paige. A Canonical Scheme for Model Composition. En *Model Driven Architecture. Foundations and Applications*, volumen 4066 de *Lecture Notes in Computer Science*, páginas 346–360. Springer, 2006.

[52] J. Bézivin, E. Breton, G. Dupé, P. Valduriez. The ATL Transformation-based Model Management Framework. Informe técnico 03.08, Institut de Recherche en Informatique de Nantes. Univ. de Nantes, 2, rue de la Houssinière, Sep 2003.

[53] P. Cáceres, V. de Castro, E. Marcos. Navigation Modelling from a User Services Oriented Approach. En *Proceedings of the Conference on Advances in Information Systems (CAISE'04)*, páginas 150–160. Springer-Verlag Lecture Notes in Computer Science, 2004.

[54] C. Cachero, N. Koch. Conceptual Navigation Analysis: A device and Platform Independent Navigation Specification. En *Proceedings of the Second International Workshop on Web-oriented Software Technology (IWWOST'02)*, páginas 21–32. CYTED, junio 2002.

[55] S. Casteleyn, F. Daniel, P. Dolog, M. Matera. *Engineering Web Applications*. Data-Centric Systems and Applications. Springer, 2009.

[56] S. Casteleyn, W. V. Woensel, G.-J. Houben. A Semantics-based Aspect-Oriented Approach to Adaptation in Web Engineering. En *Proceedings of 18th conference on Hypertext and hypermedia (HT07)*, páginas 189–198. ACM, 2007.

[57] P. Cáceres, V. de Castro, J. M. Vara, E. Marcos. Model Transformations for Hypertext Modeling on Web Information Systems. En *Proceedings of The Symposium on Applied Computing (SAC'06)*, abril 2006.

[58] S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, M. Matera. *Designing Data-Intensive Web Applications*. Elsevier Science, 2003.

[59] R. Chitchyan, A. Rashid, P. Sawyer, A. Garcia, M. P. Alarcon, J. Bakker, B. Tekinerdogan, S. Clarke, A. Jackson. Survey of Aspect-Oriented Analysis and Design Approaches. Informe técnico AOSD-Europe-ULANC-9, AOSD-Europe, mayo 2005.

[60] S. Clarke, E. Baniassad. *Aspect-Oriented Analysis and Design: The Theme Approach*. Addison-Wesley Professional, 2005.

[61] T. Cleenewerck, J. Noye. Editorial Domain Specific Aspect Languages. *IET Software*, 3(3):165–166, junio 2009.

[62] T. Cleenewerck, J. Noyé, J. Fabry, A.-F. Lemeur, E. Tanter. Summary of the Third Workshop on Domain-Specific Aspect Languages. En *Proceedings of the 2008 AOSD workshop on Domain-specific Aspect Languages*, DSAL'08, páginas 1–5, New York, NY, USA, 2008. ACM.

[63] W. Coelho, G. C. Murphy. Modeling Aspects: An Implementation-Driven Approach. En *Proceedings of the Workshop on Best Practices for Model-Driven Software Development (held with OOPSLA 2004),*. ACM Press, 2004.

[64] A. Colyer. Towards Widespread Adoption of AOSD. En *AOSD Workshop on Commercialization of AOSD Technology*, marzo 2003.

[65] Compuware. OptimalJ. `http://www.compuware.com/products/optimalj/`, 2007.

[66] J. Conallen. *Building Web Applications with UML*. Addison-Wesley, 2 edición, 2003.

[67] J. M. Conejero, K. G. van den Berg, J. Hernández. Un Marco Conceptual para la Formalización de Crosscutting en Desarrollos Orientados a Aspectos. *IEEE Latin America Transactions*, 5(4), julio 2007.

[68] M.-T. Consortium. Assessment of the Model Driven Technologies-Foundations and Key Technologies. Informe técnico IST-2001-37785, MODA-TEL Consortium, 2002.

[69] S. Cook. Domain-Specific Modeling and Model Driven Architecture. *MDA Journal*, 2004.

[70] S. Cook, G. Jones, S. Kent, A. C. Wills. *Domain-specific Development with Visual Studio DSL Tools*. Microsoft .NET Development Series. Addisson-Wesley, 2007.

[71] IBM Corporation. HyperJ. `http://www.research.ibm.com/hyperspace/HyperJ/HyperJ.htm`, 2002.

[72] M. F. Costabile, D. Fogli, A. Marcante. Supporting Interaction and Co-evolution of Users and Systems. En *Advanced Visual Interfaces - AVI (2006)*, 2006.

[73] T. Cottenier, A. van den Berg, T. Elrad. Modeling Aspect-Oriented Compositions. En *Proceedings of the Satellite Events at 8th International Conference on Model Driven Engineering Languages and Systems*, número 3844, páginas 100–109. Springer-Verlag Lecture Notes in Computer Science, abril 2005.

[74] CREST. The Coral Metamodeling Toolkit. `http://crestwiki.abo.fi/confluence/display/CRL/Home`.

[75] G. Csertán, G. Huszerl, I. Majzik, Z. Pap, A. Pataricza, D. Varró. VIATRA - Visual Automated Transformations for Formal Verification and Validation of UML Models. En *Proceedings of 17th International Conference on Automated Software Engineering (2002)*, páginas 267–270, 2002.

[76] J. M. Cueva Lovelle, B. Martín González Rodríguez, L. Joyanes Aguilar, J. E. L. Gayo, M. del Puerto Paule Ruíz, editores. *Web Engineering, International Conference, ICWE 2003, Oviedo, Spain, July 14-18, 2003, Proceedings*, volumen 2722 de *Lecture Notes in Computer Science*. Springer, 2003.

[77] K. Czarnecki, U. W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, Boston, 2000.

[78] K. Czarnecki, S. Helsen. Feature-based Survey of Model Transformation Approaches. *IBM Systems Journal*, 45(3), 2006.

[79] J. de Lara. AToM3: A Tool for Multi-formalism and Meta-Modelling. `http://moncs.cs.mcgill.ca/people/jlara/ATOM3.html`, 2006.

[80] O. De Troyer, T. Decruyenaere. Conceptual Modelling of Web Sites for End-Users. *WWW Journal*, 3(1):27–42, 2000.

[81] D. Di Ruscio, H. Muccini, A. Pierantonio. A Data Modeling Approach to Web Application Synthesis. *International Journal of Web Engineering and Technology*, 1(3):320–337, 2004.

[82] O. Díaz, A. Irastorza, M. Azanza, F. M. Villoria. Modeling Portlet Aggregation Through Statecharts. En *Proceedings of the Conference on Web Information Systems (WISE 06)*, volumen 4255, páginas 265–276. Springer-Verlag Lecture Notes in Computer Science, 2006.

[83] M. Didonet del Fabro. *Metadata Management Using Model Weaving and Model Transformation*. PhD thesis, Faculté Des Sciences Et Des Techniques. Université De Nantes, 2007.

[84] M. Didonet del Fabro, J. Bézivin, F. Jouault, E. Breton, G. Gueltas. AMW: A Generic Model Weaver. En *Proceedings of the Premières Journées sur lÍngénierie Dirigée par les Modèles*, julio 2005.

[85] M. Didonet del Fabro, J. Bézivin, P. Valduriez. Weaving Models with the Eclipse AMW Plugin. En *Eclipse Modeling Symposium, Eclipse Summit Europe 2006*, octubre 2006. `http://www.eclipsecon.org/summiteurope2006/presentations/ESE2006-EclipseModelingSymposium2_WeavingModels.pdf`.

[86] E. W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, Englewood Cliffs, New Jersey, 1976.

[87] P. Dolog, M. Bieliková. Hypermedia Modelling Using UML. En *Proceedings of ISM 2002*, 2002.

[88] P. Dolog, W. Nejdl. Using UML and XMI for Generating Adaptive Navigation Sequences in Web-Based Systems. En *Proceedings of «UML» 2003 - Sixth International Conference on the Unified Modeling Language: Modeling Languages and Applic*, número 2863. Springer-Verlag Lecture Notes in Computer Science, octubre 2003.

[89] K. Donald, E. Vervaet. The Spring Web Flow Home Page. `http://opensource.atlassian.com/confluence/spring/display/WEBFLOW/Home`.

[90] R. Douence, P. Fradet, M. Südholt. A Framework for the Detection and Resolution of Aspect Interactions. En *GPCE02: Proceedings of 1st ACM SIGPLAN/-SIGSOFT conference on Generative Programming and Component Engineering*, páginas 173–188, London, UK, 2002. Springer-Verlag Lecture Notes in Computer Science.

[91] D. Dougherty, A. Robbins. *sed & awk*. O'Reilly & Associates, 2 edición, 1997.

[92] K. Duddy, A. Gerber, K. Raymond. Eclipse Modelling Framework (EMF) Import/Export from MOF / JMI. Informe técnico, Queensland University of Technology, Nov 2003.

[93] Eclipse AMW Project. ATLAS Model Weaver Project. `http://www.eclipse.org/gmt/amw/`, 2010.

[94] Eclipse ATL Project. ATLAS Tranformation Language. `http://www.eclipse.org/m2m/atl/`, 2010.

[95] Eclipse EMP Project. Eclipse Modeling Project. `http://www.eclipse.org/modeling/`, 2007.

[96] Eclipse Epsilon Project. Epsilon Merging Language. `http://www.eclipse.org/gmt/epsilon/doc/eml/`, 2010.

[97] Eclipse Epsilon Project. Epsilon Project. `http://www.eclipse.org/gmt/epsilon/`, 2010.

[98] Eclipse GMT Project. Generative Modeling Technologies (GMT). `http://eclipse.org/gmt/`, 2007.

[99] Eclipse M2M. QVT Operational. `http://www.eclipse.org/m2m/qvto/doc/`.

[100] S. Efftinge, C. Kadura. OpenArchitectureWare 4.1. Xpand Language Reference. `http://www.eclipse.org/gmt/oaw/doc/4.1/r20$_$xPandReference.pdf`.

[101] Elibel. SmartQVT. `http://smartqvt.elibel.tm.fr/`.

[102] T. Elrad, R. E. Filman, A. Bader. Aspect-Oriented Programming. *Communications ACM*, 44(10):29–32, octubre 2001.

[103] E. A. Emerson. *Handbook of Theoretical Computer Science: Formal Models and Semantics*, capítulo Temporal and Modal Logic, páginas 995–1072. MIT Press, 1990.

[104] M. Emerson, J. Sztipanovits. Implementing a MOF-Based Metamodeling Environment Using Graph Transformations. En *Proceedings of 4th OOPSLA Workshop on Domain-Specific Modeling*, páginas 83–92, octubre 2004.

[105] M. Emrich. Generative Programming Using Frame Technology. Diploma thesis, University of Applied Sciences. Department of Computer Science and Microsystems Engineering, Kaiserslautern. Germany, 2003.

[106] K. D. Engel, R. F. Paige, D. S. Kolovos. Using a Model Merging Language for Reconciling Model Versions. En *ECMDA-FA*, volumen 4066 de *Lecture Notes in Computer Science*, páginas 143–157. Springer, 2006.

[107] M. J. Escalona, M. Mejías, J. Torres, A. M. Reina. The NDT Development Process. En J. M. Cueva Lovelle y otros [76], páginas 463–467.

[108] M. J. Escalona, A. M. Reina, J. Torres, M. Mejías. The Navigational Aspect in the Requirement Specification of NDT. En *Proceedings of Iadis International Conference WWW/Internet 2004*, páginas 1237–1238. Iadis Press, 2004.

[109] M. J. Escalona, A. M. Reina, J. Torres, M.Mejías. NDT: A Methodology to Deal with the Navigation Aspect at the Requirements Phase. En *Early Aspects 2004: Aspect-Oriented Requirements Engineering and Architecture Design Workshop at OOPSLA 2004*, octubre 2004.

[110] M. J. Escalona, J. Torres, M. Mejías, A. M. Reina. NDT-Tool: A Case Tool to Deal with Requirements in Web Information Systems. En J. M. Cueva Lovelle y otros [76], páginas 212–213.

[111] J. Evermann. A Meta-level Specification and Profile for AspectJ in UML. En *AOM'07: Proceedings of the 10th international workshop on Aspect-oriented modeling*, páginas 21–27, New York, NY, USA, 2007. ACM Press.

[112] J. Fabry, Éric Tanter, T. D'Hondt. KALA: Kernel Aspect Language for Advanced Transactions. *Science of Computer Programming*, 71(3):165 – 180, 2008.

[113] X. Fang, C. Holsapple. Impacts of Navigation Structure, Task Complexity, and Users'Domain Knowledge on Web Site Usability. An Empirical Study. *Information Systems Frontiers*, páginas 1–17, 2010.

[114] K. Farias, A. Garcia, J. Whittle. Assesing the Impact of Aspects on Model Composition Effort. En *Proceedings of the International Conference on Aspect-Oriented Software Development (AOSD'10)*, páginas 73–84. ACM, marzo 2010.

[115] M. C. Ferreira de Oliveira, M. A. Santos Turine, P. Masiero. A Statechart-Based Model for Hypermedia Applications. *ACM Transactions on Information Systems*, 19(1):28–52, enero 2001.

[116] R. E. Filman. Achieving Ilities. En *OMG-DARPA Workshop on Compositional Software Architectures*, enero 1998.

[117] R. E. Filman. What Is Aspect-Oriented Programming, Revisited. En *Workshop on Advanced Separation of Concerns (ECOOP 2001)*, junio 2001.

[118] R. E. Filman. A Bibliography of Aspect-Oriented Software Development. Version 2.0. Informe técnico, Research Institute for Advanced Computer Science, noviembre 2005.

[119] R. E. Filman, T. Elrad, S. Clarke, M. Akşit, editores. *Aspect-Oriented Software Development*. Addison-Wesley, Boston, octubre 2004.

[120] D. Fiorito, R. Dalton. Creating a Consistent Enterprise Web Navigation Solution. `http://iasummit.org/2004/finalpapers/FioritoDalton_Handout_or__final__paper.ppt`, febrero 2004.

[121] F. Fleurey, B. Baudry, R. France, S. Ghosh. A Generic Approach for Automatic Model Composition. En *MODELS 2007 Workshops*, número 5002, páginas 7–15. Springer-Verlag Lecture Notes in Computer Science, 2007.

[122] F. Fondement. *Concrete Syntax Definition for Modeling Languages*. PhD thesis, École Polytechnique Fédérale De Lausanne, noviembre 2007.

[123] J. Fons, V. Pelechano, Óscar Pastor, P. Valderas, V. Torres. *Web Engineering: Modelling and Implementing Web Applications*, capítulo 5. Applying the OOWS Model-Driven Approach for Developing Web Applications. The Internet Movie Database Case Study, páginas 157–191. Springer Verlag, 2008.

[124] Stanford Center for Biomedical Informatics Research. Protégé Home Page. `http://protege.stanford.edu/`.

[125] The Eclipse Foundation. MoDisco. `http://wiki.eclipse.org/MoDisco`, Sept 2010.

[126] The Eclipse Foundation. Eugenia. `http://www.eclipse.org/gmt/epsilon/doc/eugenia/`, enero 2011.

[127] The Eclipse Foundation. Graphical Modelling Project. website:`http://www.eclipse.org/modeling/gmp/`, enero 2011.

[128] M. Fowler. *Domain-Specific Languages*. Signature. Addison-Wesley, 1 edición, 2010.

[129] R. France, J. M. Bieman. Multi-View Software Evolution: A UML-based Framework for Evolving Object-Oriented Software. En *Proceedings of the International Conference on Software Mainteinance (ICSM 2001)*, noviembre 2001.

[130] R. France, D. K. Kim, S. Ghosh, E. Song. A UML-based Specification Technique. *IEEE Transaction Software Engineering*, 30(3):193–206, 2004.

[131] R. France, I. Ray, G. Georg, S. Ghosh. Aspect-Oriented Approach to Early Design Modeling. *IEE Software*, 151:173–186, junio 2004.

[132] R. Z. Frantz, A. M. Reina, R. Corchuelo. A Domain-Specific Language to Design Enterprise Application Integration Solutions. *International Journal of Cooperative Information Systems*, 20(2):143–176, 2011.

[133] P. Fraternali, P. Paolini. A Conceptual Model and a Tool Environment for Developing More Scalable and Dynamic Web Applications. En *Proceedings of the Conference On Extended Database Technology (EDBT'98)*, páginas 421–435, marzo 1998.

[134] L. Fuentes, P. Sánchez. Designing and Weaving Aspect-Oriented Executable UML models. *Journal of Object Technology - Special Issue on Aspect-Oriented Modelling*, 6(7):109–136, agosto 2007.

[135] B. Ganter, R. Wille. *Formal Concept Analysis*. Springer Verlag, 1996.

[136] F. Garzotto, L. Mainetti, P. Paolini. Hypermedia Design, Analysis and Evaluation Issues. *Communications of the ACM*, 38(8):74–86, agosto 1995.

[137] G. Georg, R. Reddy, R. B. France. Specifying Cross-Cutting Requirement Concerns. En *UML*, volumen 3273 de *Springer-Verlag Lecture Notes in Computer Science*, páginas 113–127. Springer-Verlag, 2004.

[138] A. Gerber, K. Raymond. MOF to EMF: There and Back Again. En *Proceedings of the Eclipse Technology Exchange Workshop, OOPSLA 2003*, páginas 6–70, octubre 2003.

[139] A. Ginige. Designing Web Applications to Support Co-Evolution. En *2nd International Conference on Web Engineering and Applications 2007*. Narosa Publishing House, 2007.

[140] A. Ginige, B. D. Silva. CBEADS©. A Framework to Support Meta-Design Paradigm. En *Proceedings of the Human and Computer Interaction Conference (HCI 2007)*, número 4554, páginas 107–116. Springer-Verlag Lecture Notes in Computer Science, 2007.

[141] J. Ginzburg, D. Distante, G. Rossi, M. Urbieta. Oblivious Integration of Volatile Functionality in Web Application Interfaces. *J. Web Eng.*, 8:25–47, marzo 2009.

[142] J. Ginzburg, G. Rossi, M. Urbieta, D. Distante. Transparent Interface Composition in Web Applications. En *Proceedings of the International Conference on Web Engineering (ICWE'07)*, páginas 152–166. Springer-Verlag Lecture Notes in Computer Science, 2007.

[143] M. T. Gómez-López, A. M. Reina, R. M. Gasca. Model-Driven Engineering for Constraint Database Query Evaluation. En *Proceedings of the Workshop on Model-Driven Engineering, Logic and Optimization: Friends or Foes? (MELO'11)*, 2011.

[144] T. Goldschmidt, S. Becker, A. Uhl. Classification of Concrete Textual Syntax Mapping Approaches. En *Model Driven Architecture. Foundations and Applications*, volumen 5095 de *Lecture Notes in Computer Science*, páginas 169–184. Springer, 2008.

[145] J. Gómez, C. Cachero. OO-H Method: Extending UML to Model Web Interfaces. En *Information Modeling for Internet Applications*. Idea Group Inc., 2003.

[146] S. Gordillo, G. Rossi, A. Moreira, J. Araújo, C. Vairetti, M. Urbieta. Modeling and Composing Navigational Concerns in Web Applications. Requirements and Design Issues. En *Proceedings of the Fourth Latin American Web Congress, 2006. LA-Web'06*, páginas 25–31, octubre 2006.

[147] S. Gordillo, G. Rossi, D. Schwabe. Separation of Structural Concerns in Physical Hypermedia Models. En *Proceedings of the Conference on Advanced Information Systems Engineering (CAISE'05)*, páginas 446–459. Springer-Verlag Lecture Notes in Computer Science, 2005.

[148] E. A. Gorshkova, B. A. Novikov. Use of Statechart Diagrams for Modeling of Hypertext. *Programming and Computer Software*, 30(1):47–51, 2004.

[149] C. Grannell. *The Essential Guide to CSS and HTML Web Design*, capítulo 5. Using Links and Creating Navigation, páginas 147–232. friendsof. APress, 2007.

[150] J. Gray, T. Bapty, S. Neema, D. Schmidt, A. Gokhale, B. Natarajan. An Approach for Supporting Aspect-Oriented Domain Modeling. En *Generative Programming and Component Engineering*, volumen 2830 de *Lecture Notes in Computer Science*, páginas 151–168. Springer, 2003.

[151] J. Greenfield, K. Short, S. Cook, S. Kent. *Software Factories. Assembling Applications with Patterns, Models, Frameworks and Tools*. Wiley Publishing, Inc., 2004.

[152] D. Groenewegen, Z. Hemel, E. Visser. Separation of Concerns and Linguistic Integration in WebDSL. *IEEE Software*, 27:31–37, setiembre 2010.

[153] D. Groenewegen, E. Visser. Declarative Access Control for WebDSL: Combining Language Integration and Separation of Concerns. En *Eighth International Conference on Web Engineering, 2008. ICWE'08*, páginas 175 –188, julio 2008.

[154] D. Groenewegen, E. Visser. Integration of Data Validation and User Interface Concerns in a DSL for Web Applications. *Software and Systems Modeling*, páginas 1–18, 2010.

[155] I. Groher, S. Schulze. Generating Aspect Code from UML Models. En O. Aldawud y otros [10].

[156] R. C. Gronback. *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*. Addison-Wesley Professional, 2009.

[157] Atlas Group. The AMMA Platform. `http://www.sciences.univ-nantes.fr/lina/atl/AMMAROOT/`.

[158] Fujaba Development Group. Fujaba Tool Suite Home Page. `http://wwwcs.uni-paderborn.de/cs/fujaba/projects/tgg/index.html`, 2006.

[159] ISIS Research Group. Moment Home Page. `http://moment.dsic.upv.es`.

[160] Triskell Group. Kermeta. Triskell Metamodelling Kernel. `http://www.kermeta.org/`, 2007.

[161] The Distributed Group. TDG Scholar. `http://scholar.tdg-seville.info/`, 2009.

[162] TU Dresden: Software Technology Group. EMFText. `http://emftext.org`.

[163] Web Engineering Group. ArgoUWE Home Page. `http://www.pst.ifi.lmu.de/projekte/uwe/toolargoUWE.html`, 2008.

[164] J. Grundy. Aspect-oriented Requirements Engineering for Component-based Software Systems. En *4th IEEE International Symposium on Requirements Engineering*, páginas 84–91. IEEE Computer Society, 1999.

[165] J. Grundy. Multi-Perspective Specification, Design And Implementation Of Software Components Using Aspects. *International Journal of Software Engineering and Knowledge Engineering*, 10(6), 2000.

[166] F. Gutiérrez, F. Durán, E. Pimentel. *Programación Orientada a Objetos con Java*, capítulo 1. Evolución de los Lenguajes de Programación. Ed. Thomson Paraninfo, 2007.

[167] N. Guyomar. MoDisco/Components/JSP/Documentation. `http://wiki.eclipse.org/MoDisco/Components/JSP/Documentation/0.9`, setiembre 2010.

[168] N. Guyomar. MoDisco/Components/XML/Documentation. `http://wiki.eclipse.org/MoDisco/Components/XML`, setiembre 2010.

[169] B. Hailpern, P. Tarr. Model-driven Development: The Good, the Bad, and the Ugly. *IBM Systems Journals*, 45(3):451–462, 2006.

[170] M. Han, C. Baufmeister. Separation of Navigation Routing Code in J2EE Web Applications. En *Proceedings of 5th International Conference on Web Engineering (ICWE 2005)*, volumen 3579, páginas 406–416. Springer-Verlag Lecture Notes in Computer Science, 2005.

[171] M. Han, C. Hofmeister. Separating and Representing Navigation Concerns in Web Applications. Informe técnico 04-004, Lehigh University, 2004.

[172] M. Han, C. Hofmeister. Modeling Request Routing in Web Applications. *Eighth IEEE International Symposium on Web Site Evolution (WSE'06)*, 0:103–110, 2006.

[173] Y. Han, G. Kniesel, A. B. Cremers. A Meta Model and Modeling Notation for AspectJ. En O. Aldawud y otros [9].

[174] W. Harrison, H. Ossher. Subject-Oriented Programming—A Critique of Pure Objects. En *Proceedings of the 1993 Conference on Object-Oriented Programming Systems, Languages, and Applications*, páginas 411–428, setiembre 1993.

[175] W. Harrison, H. Ossher, P. Tarr. Asymmetrically vs. Symmetrically Organized Paradigms for Software Composition. En *SPLAT: Software engineering Properties of Languages for Aspect Technologies*, marzo 2003.

[176] L. Hart, P. Emery, B. Colomb, K. Raymond, S. Taraporewalla, D. Chang, Y. Ye, E. Kendall, M. Dutra. OWL Full and UML 2.0 Compared. Informe técnico, 2004.

[177] J. Heering, P. R. H. Hendriks, P. Klint, J. Rekers. The Syntax Definition Formalism SDF (Reference Manual). *SIGPLAN Not.*, 24:43–75, noviembre 1989.

[178] F. Heidenreich, J. Johannes, S. Karol, M. Seifert, C. Wende. Derivation and Refinement of Textual Syntax for Models. En *Model Driven Architecture. Foundations and Applications*, volumen 5562 de *Lecture Notes in Computer Science*, páginas 114–129. Springer, 2009.

[179] Z. Hemel, R. Verhaaf, E. Visser. WebWorkFlow: An Object-Oriented Workflow Modeling Language for Web Applications. En *Model Driven Engineering Languages and Systems*, volumen 5301 de *Lecture Notes in Computer Science*, páginas 113–127. Springer, 2008.

[180] A. Hen-Tov, D. H. Lorenz, A. Pinhasi, L. Schachter. ModelTalk: When Everything Is a Domain-Specific Language. *IEEE Software*, 26(4):39–46, 2009.

[181] R. Hennicker, N. Koch. A UML-based Methodology for Hypermedia Design. En *Proceedings of the Unified Modeling Language Conference (UML 2000)*, número 1939, páginas 410–424. Springer-Verlag Lecture Notes in Computer Science, 2000.

[182] J. L. Herrero, F. Sánchez, F. Lucio, M. Toro. Introducing Separation of Aspects at Design Time. En *Workshop on Aspects and Dimensions of Concerns (ECOOP 2000)*, junio 2000.

[183] S. Herrmann. Composable Designs with UFA. En AOSD-UML'02 [17].

[184] W.-M. Ho, F. Pennaneac'h, N. Plouzeau. UMLAUT: A Framework for Weaving UML-Based Aspect-Oriented Designs. En *Proceedings of 33rd International Conference on Technology of Object-Oriented Languages (TOOLS'00)*, páginas 324–334. IEEE Computer Society, 2000.

[185] G. Hohpe, B. Woolf. *Enterprise Integration Patterns - Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley, 2003.

[186] I. Horrocks. *Constructing the User Interface with Statecharts*. Adisson-Wesley, 1999.

[187] G. Houben, F. Frasincar, P.Barna, R. Vdovjak. Modeling User Input and Hypermedia Dynamics in Hera. En *Proceedings of 4th International Conference on Web Engineering (ICWE 2004)*, volumen 3140, páginas 60–73. Springer-Verlag Lecture Notes in Computer Science, 2004.

[188] A. Hovsepyan, S. B. Y. B. W. Joosen. Specifying and Composing Concerns Expressed in Domain-Specific Modeling Languages. En *Objects, Components, Models and Patterns*, volumen 33 de *Lecture Notes in Business Information Processing*, páginas 116–135. Springer Berlin Heidelberg, 2009.

[189] A. Hovsepyan, S. Van Baelen, Y. Berbers, W. Joosen. Generic Reusable Concern Compositions. En *Model Driven Architecture. Foundations and Applications*, volumen 5095 de *Lecture Notes in Computer Science*, páginas 231–245. Springer, 2008.

[190] A. Hovsepyan, S. Van Baelen, Y. Berbers, W. Joosen. Specifying and Composing Concerns Expressed in Domain-Specific Modeling Languages. En *47th International Conference on Objects, Models, Components, Patterns*, volumen 33, páginas 116–135. Springer, junio 2009.

[191] Ikv++ Technologies Ag. Medini QVT. `http://projects.ikv.de/qvt`.

[192] Inspera. Information Architecture and Navigation Design. `http://www.inspera.com/servlets/dispatcher?siteNodeId=585229&languageId=2`, 2009.

[193] J. Irwin, J.-M. Loingtier, J. R. Gilbert, G. Kiczales, J. Lamping, A. Mendhekar, T. Shpeisman. Aspect-Oriented Programming of Sparse Matrix Code. En *International Scientific Computing in Object-Oriented Parallel Environments (ISCOPE)*, volumen 1343 de *LNCS*. Springer-Verlag, 1997.

[194] T. Isakowitz, A. Kamis, M. Koufaris. Reconciling Top-Down and Bottom-Up Design Approaches in RMM. *DATA BASE*, 28(4):58–67, 1998.

[195] ISIS. The Generic Modeling Environment. `http://www.isis.vanderbilt.edu/Projects/gme/index.html`.

[196] ISIS. Model-Integrated Computing. `http://www.isis.vanderbilt.edu/research/research.html`.

[197] J. Nanard, G. Rossi, M. Nanard, S. Gordillo, L. Pérez. Concern-Sensitive Navigation: Improving Navigation in Web Software through Separation of Concerns. En *Proceedings of the Conference on Advanced Information Systems Engineering (CAISE'08)*, volumen 5074, páginas 420–434. Springer-Verlag Lecture Notes in Computer Science, 2008.

[198] I. Jacobson, G. Booch, J. Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, 1999.

[199] C. Jeanneret. An Analysis of Model Composition Approaches. Master's thesis, Colorado State University and École Polytechnique Fédérale de Lausanne, 2008.

[200] C. Jeanneret, R. France, B. Baudry. A Reference Process for Model Composition. En *Proceedings of AOM workshop in conjunction with AOSD'08*, marzo 2008.

[201] R. Johnson, J. Hoeller, A. Arendsen, C. Sampaleanu, R. Harrop, T. Risberg, D. Davison, D. Kopylenko, M. Pollack, T. Templier, E. Vervaet, P. Tung, B. Hale, A. Colyer, J. Lewis, C. Leau, R. Evans. The Spring Framework - Reference Documentation. `http://static.springframework.org/spring/docs/2.0.x/spring-reference.pdf`.

[202] F. Jouault, J. Bézivin. KM3: A DSL for Metamodel Specification. En *Proceedings of Formal Methods for Open Object-Based Distributed Systems (FMOODS'06)*, número 4037, páginas 171–185. Springer-Verlag Lecture Notes in Computer Science, 2006.

[203] F. Jouault, J. Bézivin, I. Kurtev. TCS: a DSL for the Specification of Textual Concrete Syntaxes in Model Engineering. En *GPCE'06: Proceedings of 5th international conference on Generative Programming and Component Engineering*, páginas 249–254. ACM Press, 2006.

[204] F. Jouault, I. Kurtev. On the Architectural Alignment of ATL and QVT. En *Proceedings of the 2006 ACM Symposium on Applied Computing (SAC'06)*, páginas 1188–1195. ACM Press, 2006.

[205] J. U. Júnior, R. D. Penteado, V. V. de Camargo. An Overview and an Empirical Evaluation of UML-AOF: An UML Profile for Aspect-Oriented Frameworks. En *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC'10, páginas 2289–2296, New York, NY, USA, 2010. ACM.

[206] J. Kalbach. *Designing Web Navigation.* O'Reilly Media, Inc., 2007.

[207] M. Kandé, O. Aldawud, G. Booch, B. Harrison, editores. *Second International Workshop on Aspect-Oriented Modeling with UML (<<UML>>2002)*, setiembre 2002.

[208] M. M. Kandé, J. Kienzle, A. Strohmeier. From AOP to UML—A Bottom-Up Approach. En AOSD-UML'02 [17].

[209] G. M. Kapitsaki, D. A. Kateros, G. N. Prezerakos, I. S. Venieris. Model-driven Development of Composite Context-aware Web Applications. *Information and Software Technology*, 51(8):1244 – 1260, 2009.

[210] M. Katara, S. Katz. Architectural Views of Aspects. En M. Akşit [4], páginas 1–10.

[211] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, J. Irwin. Aspect-Oriented Programming. En *11th Europeen Conference on Object-Oriented Programming*, volumen 1241 de *LNCS*, páginas 220–242. Springer Verlag, 1997.

[212] J. Kienzle, W. Al Abed, F. Fleurey, J.-M. Jézéquel, J. Klein. Aspect-Oriented Design with Reusable Aspect Models. En *Transactions on Aspect-Oriented Software Development VII*, volumen 6210 de *Lecture Notes in Computer Science*, páginas 272–320. Springer, 2010.

[213] J. Kienzle, J. Gray, D. Stein, W. Cazzola, O. Aldawud, T. Elrad. 11th International Workshop on Aspect-Oriented Modeling. En *Proceedings of the 11th International Workshop on Aspect-Oriented Modeling held in conjunction with the International Conference on Model-Driven Engineering, Languages, and Systems (MODELS)*, 2008.

[214] J. Klein, L. Hélouet, J.-M. Jézéquel. Semantic-based Weaving of Scenarios. En *Proceedings of 5th International Conference on Aspect-Oriented Software Development (AOSD'06)*, Bonn, Germany, mar 2006. ACM.

[215] A. Kleppe, J. Warner, W. Best. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison Wesley, 2003.

[216] A. Knapp, N. Koch, F. Moser, G. Zhang. ArgoUWE: A Case Tool for Web Applications. En *Proceedings of the First International Workshop on Engineering Methods to Support Information Systems Evolution (EMSISE'03)*, 2003.

[217] N. Koch. *Software Engieneering for Adaptive Hipermedia System: Reference Model, Modeling Techniques and Development Process*. PhD thesis, Ludwig-Maximilian-Universitat Munchen, 2001.

[218] N. Koch. Transformations Techniques in the Model-driven Development Process of UWE. En *Proceedings of 2nd Model-Driven Web Engineering Workshop (MDWE'06)*, volumen 155. ACM, 2006.

[219] N. Koch, A. Knapp, G. Zhang, H. Baumeister. *Web Engineering: Modelling and Implementing Web Applications*, capítulo 7. UML-based Web Engineering. An Approach Based on Standards, páginas 157–191. Springer Verlag, 2008.

[220] S. Kojarski, D. H. Lorenz. Pluggable AOP: Designing Aspect Mechanisms for Third-party Composition. En *Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, OOPSLA'05, páginas 247–263, New York, NY, USA, 2005. ACM.

[221] D. Kolovos, R. Paige, F. Polack. Merging Models with the Epsilon Merging Language (EML). En *Model Driven Engineering Languages and Systems*, volumen 4199 de *Lecture Notes in Computer Science*, páginas 215–229. Springer, 2006.

[222] D. Kolovos, R. F. Paige, F. A. Polack. Detecting and Repairing Inconsistencies Across Heterogeneous Models. En *Proceedings of 2008 International Conference*

on Software Testing, Verification, and Validation (ICST'08), páginas 356–364. IEEE, 2008.

[223] D. Kolovos, L. Rose, S. Abid, R. Paige, F. Polack, G. Botterweck. Taming EMF and GMF Using Model Transformation. En *Model Driven Engineering Languages and Systems*, volumen 6394 de *Lecture Notes in Computer Science*, páginas 211–225. Springer, 2010.

[224] A. Kraus. *Model Driven Software Engineering for Web Applications*. PhD thesis, Ludwig-Maximilians-Universität, München, 2007.

[225] A. Kraus, A. Knapp, N. Koch. Model-Driven Generation of Web Applications in UWE. En *In Proceedings of 3rd Model-Driven Web Engineering Workshop (MDWE'07)*, volumen 261. CEUR-WS, 2007.

[226] C. Kroib, N. Koch. UWE Metamodel and Profile: User Guide and Reference. Informe técnico 0802, Ludwig-Maximilians-Universität München (LMU), febrero 2008.

[227] V. Kulkarni, S. Reddy. Addressing Separation of Concerns in MDD. *IEEE Software*, 20(5):64–69, setiembre 2003.

[228] I. Kurtev, J. Bézivin, M. Aksit. Technological Spaces: An Initial Appraisal. En *International Federated Conferences (DOA, ODBASE, CoopIS), Industrial Track*, 2002.

[229] I. Kurtev, J. Bézivin, F. Jouault, P. Valduriez. Model-based DSL Frameworks. En *OOPSLA'06: Companion to the 21st ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications*, páginas 602–616, New York, NY, USA, 2006. ACM.

[230] P. Lahire, B. Morin, G. Vanwormhoudt, A. Gaignard, O. Barais, J.-M. Jézéquel. Introducing Variability into Aspect-Oriented Modeling Approaches. En *Model Driven Engineering Languages and Systems*, volumen 4735 de *Lecture Notes in Computer Science*, páginas 498–513. Springer, 2007.

[231] K. Leung, L. Hui, S. Yiu, R. Tang. Modelling Web Navigation by Statechart. En *Proceedings of 24th Annual International Computer Software and Applications Conference (COMPSAC 2000)*, páginas 41–47. IEEE, octubre 2000.

[232] K. Lieberherr, D. Orleans, J. Ovlinger. Aspect-Oriented Programming with Adaptive Methods. *Communications ACM*, 44(10):39–41, octubre 2001.

[233] K. J. Lieberherr. *Adaptive Object-Oriented Software: the Demeter Method with Propagation Patterns*. PWS Publishing Company, Boston, 1996.

[234] K. J. Lieberherr, D. Orleans. Preventive Program Maintenance in Demeter/Java (Research demonstration). En *International Conference on Software Engineering*, páginas 604–605, Boston, MA, 1997. ACM Press.

[235] F. Lima, D. Schwabe. Modeling Applications for the Semantic Web. En *Proceedings of International Conference on Web Engineering (ICWE'03)*, número 2722, páginas 417–426. Springer-Verlag Lecture Notes in Computer Science, 2003.

[236] J. M. Lions, D. Simoneau, G. Pitette, I. Moussa. Extending OpenTool/UML Using Metamodeling: An Aspect-Oriented Programming Case Study. En M. Kandé y otros [207].

[237] C. V. Lopes. *D: A Language Framework for Distributed Programming.* PhD thesis, College of Computer Science, Northeastern University, 1997.

[238] C. V. Lopes. AOP: A Historical Perspective (What's in a Name?). En R. E. Filman y otros [119], páginas 97–122.

[239] C. V. Lopes, S. K. Bajracharya. An Analysis Of Modularity In Aspect Oriented Design. En *Proceedings of 4th International Conference on Aspect-Oriented Software Development (AOSD 2005)*, páginas 15–26. ACM Press, marzo 2005.

[240] M. Mahoney, A. Bader, T. Elrad, O. Aldawud. Using Aspects to Abstract and Modularize Statecharts. En O. Aldawud y otros [9].

[241] H. Masuhara, G. Kiczales. Modular Crosscutting in Aspect-oriented Mechanisms. En *ECOOP 2003—Object-Oriented Programming, 17th European Conference*, volumen 2743 de *lncs*, páginas 2–28, Berlin, julio 2003. Springer-Verlag.

[242] K. L. McMillan. *Symbolic Model Checking.* Academic Publishers, 1993.

[243] A. McNeile, E. Roubtsova. Aspect-Oriented Development Using Protocol Modeling. En *Transactions on Aspect-Oriented Software Development VII*, volumen 6210 de *Lecture Notes in Computer Science*, páginas 115–150. Springer, 2010.

[244] S. Meliá, J. Gomez, N. Koch. Improving Web Design Methods with Architecture Modeling. En *Proceedings of 6th International Conference on Electronic Commerce and Web Technologies (ECWeb 2005)*, páginas 300–305. Springer-Verlag Lecture Notes in Computer Science, agosto 2005.

[245] S. Meliá, C. Cachero. An MDA Approach for the Development of Web Applications. En *Proceedings of the International Conference on Web Engineering (ICWE 2004)*, páginas 300–305. Springer-Verlag Lecture Notes in Computer Science, 2004.

[246] S. J. Mellor. A Framework for Aspect-Oriented Modeling. En O. Aldawud y otros [11].

[247] S. J. Mellor, M. Barcell. *Executable UML: A Foundation for MDA*. Addison Wesley, 2002.

[248] S. J. Mellor, A. N. Clark, T. Futagami. Model-Driven Development - Guest's Editor Introduction. *IEEE Software*, 20(5):36–41, 2003.

[249] A. Mendhekar, G. Kiczales, J. Lamping. RG: A Case-Study for Aspect-Oriented Programming. Informe técnico SPL-97-009, Palo Alto Research Center, 1997.

[250] T. Mens, P. V. Gorp. A Taxonomy of Model Transformation. En *Proceedings of the International Workshop on Graph and Model Transformation (GraMoT 2005)*, volumen 152, páginas 125–142. Electronic Notes in Theoretical Computer Science, marzo 2006.

[251] T. Mens, P. V. Gorp, D. Varró, G. Karsai. Applying a Model Transformation Taxonomy to Graph Transformation Technology. En *Proceedings of the International Workshop on Graph and Model Transformation (GraMoT 2005)*, volumen 152, páginas 143–159. Electronic Notes in Theoretical Computer Science, marzo 2006.

[252] M. Mezini, K. Ostermann. Modules for Crosscutting Models. En *8th International Conference on Reliable Software Technologies (Ada-Europe'03)*. Springer-Verlag Lecture Notes in Computer Science, junio 2003.

[253] Microsoft. Microsoft Domain-Specific Language (DSL) Tools. `http://msdn.microsoft.com/vstudio/DSLTools/`.

[254] S. Microsystems. JavaServer Pages (TM) 1.2 Specification. Informe técnico REC-xml-20081126, Eduardo Pelegrí-Llopart, setiembre 2001.

[255] M. Mohamed, M. Romdhani, K. Ghedira. MOF-EMF Alignment. En *Proceedings of the Third International Conference on Autonomic and Autonomous Systems (ICAS'07)*. IEEE Computer Society, 2007.

[256] A. Moreira, J. Araújo, A. Rashid. A Concern-oriented Requirement Engineering Model. En *Proceedings of the Conference on Advanced Information Systems Engineering (CAISE'05)*, páginas 293–308. Springer-Verlag Lecture Notes in Computer Science, 2005.

[257] A. Moreira, A. Rashid, J. Araújo. Multi-Dimensional Separation of Concerns in Requirements Engineering. En *Proceedings of the 13th IEEE International Requirements Engineerning Conference (RE 2005)*, páginas 285–296. IEEE Computer Society, agosto 2005.

[258] N. Moreno, P. Fraternali, A. Vallecillo. WebML Modelling in UML. *IET Software*, 1(3):67–80, 2007.

[259] N. Moreno, J. R. Romero, A. Vallecillo. Incorporating Cooperative Portlets in Web Application Development. En *Proceedings of 1st International Workshop on Model-Driven Web Engineering (MDWE'05)*, julio 2005.

[260] N. Moreno, A. Vallecillo. A Model-Based Approach for Integrating Third Party Systems with Web Applications. En *Proceedings of International Conference on Web Engineering (ICWE'05)*, número 3579, páginas 441–452. Springer-Verlag Lecture Notes in Computer Science, 2005.

[261] N. Moreno, A. Vallecillo. Modeling Interactions between Web Applications and Third Party Systems. En *Proceedings of the V International Workshop on Web Oriented Software Technologies (IWWOST'05)*, junio 2005.

[262] B. Morin, J. Klein, O. Barais, J.-M. Jézéquel. A Generic Weaver for Supporting Product Lines. En *Proceedings of the 13th International workshop on Early Aspects*, EA'08, páginas 11–18, New York, NY, USA, 2008. ACM.

[263] A. Muller. Reusing Functional Aspects: From Composition to Parameterization. En O. Aldawud y otros [9].

[264] P.-A. Muller. Weaving Executability into Object-Oriented Meta-Languages. En *Proceedings of the MODELS/UML 2005 Conference*, número 3713, páginas 264–278. Springer-Verlag Lecture Notes in Computer Science, 2005.

[265] P.-A. Muller, P. Studer, F. Fondement, J. Bézivin. Platform Independent Web Application Modeling and Development with Netsilon. *Software and System Modeling*, 00:1–19, 2005.

[266] J. Munnelly, S. Clarke. ALPH: A Domain-specific Language for Crosscutting Pervasive Healthcare Concerns. En *Proceedings of 2nd Workshop on Domain Specific Aspect Languages*, DSAL'07, New York, NY, USA, 2007. ACM.

[267] J. Muñoz, V. Pelechano. MDA vs. Factorías Software. En *Proceedings of the Second Workshop on Model-Driven Development, MDA and Applications (DSDM'05)*, páginas 1–10, 2005.

[268] G. Murphy, C. Schwanninger. Aspect-Oriented Programming. *IEEE Software*, 23(1):20–23, 2006.

[269] S. Murugesan. *Web Engineering: Modelling and Implementing Web Applications*, capítulo 2. Web Application Development: Challenges and the Role of Web Engineering, páginas 7–32. Human-Computer Interaction. Springer, 2008.

[270] S. Murugesan, A. Ginige. *Web Engineering. Principles and Techniques*, capítulo 1. Web Engineering: Introduction and Perspectives, páginas 1–30. Idea Publishing Group, 2005.

[271] C. Neil, C. Pons. Transformation of Models in OOHDM Using Metamodeling Techniques. En *Proceedings of the Argentine Symposium on Information Systems/Jornadas Argentinas de Informática e Investigación Operativa. JAIIO 2005*, agosto 2005.

[272] C. Nentwich, L. Capra, W. Emmerich, A. Finkelstein. xLinkIt: a Consistency Checking and Smart Link Generation Service. *ACM Transactions on Internet Technology*, 2(2):151–185, 2002.

[273] Netbeans. Metadata Repository (MDR). `http://mdr.netbeans.org/`, 2007.

[274] No Magic, Inc. MagicDraw Home Page. `http://www.magicdraw.com/`, 2008.

[275] P. Nora. *Web Application Design Patterns*, capítulo 5. Navigation. Morgan Kaufmann Publishers, 2009.

[276] M. Nouh, R. Ziarati, D. Mouheb, D. Alhadidi, M. Debbabi, L. Wang, M. Pourzandi. Aspect Weaver: A Model Transformation Approach for UML Models. En *Proceedings of the 2010 Conference of the Centre for Advanced Studies on Collaborative Research (CASCON), November 1-4, 2010, Toronto, Ontario, Canada*, páginas 139–153, noviembre 2010.

[277] D. A. Nunes, D. Schwabe. Rapid Prototyping of Web Applications Combining Domain Specific Languages and Model Driven Design. En *Proceedings of International Conference on Web Engineering (ICWE'06)*, julio 2006.

[278] B. Nuseibeh, J. Kramer, A. Finkelstein. A Framework for Expressing the Relationships between Multiple Views in Requirements Specifications. *IEEE Transactions on Software Engineering*, 20(10):760–773, octubre 1994.

[279] Computer Science Laboratory. Department of Computer Science. The Maude System. `http://maude.cs.uiuc.edu/`.

[280] J. Offutt. Quality Attributes of Web Software Applications. *IEEE Software*, 19(2):25–32, marzo 2002.

[281] Jon Oldevik. MOFScript User Guide. `http://umt-qvt.sourceforge.net/mofscript/docs/MOFScript-User-Guide.pdf`, 2006.

[282] OMG. MDA Guide Version 1.0.1, 2003.

[283] OMG. Human-Usable Textual Notation (HUTN) Specification, 2004.

[284] OMG. UML 2.0 Superstructure Specification, 2004.

[285] OMG. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. Informe técnico, OMG, Nov 2005.

[286] OMG. Meta Object Facility Specification Version 2.0. http://www.omg.org/docs/formal/06-01-01.pdf, 2006.

[287] OMG. MOF 2.0 / XMI Mapping Specification, v2.1. http://www.omg.org/technology/documents/formal/xmi.htm, 2006.

[288] OMG. MOF Models to Text Transformation Specification Language. http://www.omg.org/docs/ptc/06-11-01.pdf, 2006.

[289] S. Op de Beeck, E. Truyen, N. Boucke, F. Sanen, M. Bynens, W. Joosen. A Study of Aspect-Oriented Design Approaches. Informe técnico CW435, Department of Computer Science, Katholieke Universiteit Leuven, 2006.

[290] OpenArchitectureWare. OpenArchitectureWare. http://www.eclipse.org/gmt/oaw/.

[291] OpenArchitectureWare. OpenArchitectureWare User Guide Version 4.2. http://www.eclipse.org/gmt/oaw/doc/4.2/openArchitectureWare-42-reference.pdf.

[292] OpenArchitectureware. XPand. http://www.eclipse.org/modeling/m2t/?project=xpand.

[293] H. Ossher, P. Tarr. Multi-Dimensional Separation of Concerns using Hyperspaces. Informe técnico 21452, IBM Research Report, abril 1999.

[294] H. Ossher, P. Tarr. Multi-Dimensional Separation of Concerns and The Hyperspace Approach. En *Proceedings of the Symposium on Software Architectures and Component Technology: The State of the Art in Software Development*. Kluwer, 2000.

[295] H. Ossher, P. Tarr. The Shape of Things To Come: Using Multi-Dimensional Separation of Concerns with Hyper/J to (Re)Shape Evolving Software. *Communications ACM*, 44(10):43–50, octubre 2001.

[296] J. Palsberg, B. Patt-Shamir, K. Lieberherr. A New Approach to Compiling Adaptive Programs. *Science of Computer Programming*, 29(3):303–326, 1997.

[297] D. L. Parnas. On the Criteria To Be Used in Decomposing Systems into Modules. *Communications of the ACM*, 15(12), 1972.

[298] . Pastor. Fitting the Pieces of the Web Engineering Puzzle. En *Proceedings of the XVIII Simpósio Brasileiro de Engenharria de Software*, páginas 10–22, 2004.

[299] . Pastor, J. Gómez, E. Insfrán, V. Pelechano. The OO-method Approach for Information Systems Modeling: from Object-oriented Conceptual Modeling to Automated Programming. *Information Systems*, 26(7):507–534, 2001.

[300] . Pastor, V. Pelechano, J. Fons, S. Abrahão. Conceptual Modelling of Web Applications: the OOWS Approach. En *Web Engineering. Theory and Practice of Metrics and Measurement for Web Development*, páginas 277–302. Springer-Verlag Lecture Notes in Computer Science, 2005.

[301] F. Paterno, C. Mancini, S. Meniconi. ConcurTaskTree: A Diagrammatic Notation for Specifying Task Models. En *Proceedings of Interact 1997*, páginas 362–369. Chapman&Hall, 1997.

[302] R. Pawlak, L. Duchien, G. Florin, F. Legond-Aubry, L. Seinturier, L. Martelli. A UML Notation for Aspect-Oriented Software Design. En AOSD-UML'02 [17].

[303] I. Philippow, M. Riebisch, K. Böllert. The Hyper/UML Approach for Feature Based Software Design. En O. Aldawud y otros [11].

[304] M. Pinto, L. Fuentes, J. M. Troya. A Dynamic Component and Aspect-Oriented Platform. *Comput. J.*, 48(4):401–420, 2005.

[305] R. Popma. JET Tutorial Part 1 (Introduction to JET). `http://dev.eclipse.org/viewcvs/indextools.cgi/org.eclipse.emf/doc/org.eclipse.emf.doc/tutorials/jet1/jet_tutorial1.html`, 2005.

[306] S. Pozo, R. M. Gasca, A. M. Reina, A. J. Varela. CONFIDDENT: A Model-Driven Consistent and Non-Redundant Layer-3 Firewall ACL Design, Development and Maintenance Framework. *Journal of Systems and Software*, 2011. Aceptada para publicación.

[307] N. Prakash, S. Srivastava, S. Sabharwal. The Classification Framework for Model Transformation. *Journal of Computer Science*, 2(2):166–170, 2006.

[308] U. Priss. A Formal Concept Analysis Homepage. `http://www.upriss.org.uk/fca/fca.html`, 2007.

[309] AspectJ Project. AspectJ Home Page. `http://www.eclipse.org/aspectj/`, 2009.

[310] Eclipse Project. Eclipse Model-to-Text (M2T) Transformation Project Proposal. http://www.eclipse.org/proposals/m2t/.

[311] Eclipse EMF Project. Eclipse Modeling Framework. `http://eclipse.org/emf/`.

[312] Modelware Project. MofScript. `http://www.modelbased.net/mofscript`.

[313] A. Rashid, A. Moreira, J. Araújo. Modularisation and Composition of Aspectual Requirements. En M. Akşit [4], páginas 11–20.

[314] A. Rashid, A. Moreira, J. Araújo, P. Clements, E. Baniassad, B. Tekinerdogan. Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design. `http://www.early-aspects.net/`.

[315] Web Ratio. Web Ratio Homepage. `http://www.webratio.com`, 2008.

[316] Real Academia Española. *Diccionario de la Lengua Española.* Real Academia Española, 22 edición, 2001.

[317] A. M. Reina. Visión General de la Programación Orientada a Aspectos. Informe técnico LSI-2000-11, Departamento de Lenguajes y Sistemas Informáticos, noviembre 2000.

[318] A. M. Reina. The Spring Webflow Metamodel 1.1. Informe técnico MWACSL-TR-2008-0001, MWACSL, Avd. Reina Mercedes, s/n, febrero 2008.

[319] A. M. Reina. Surveying Navigation Modelling Approaches. *International Journal of Computer Application in Technology (IJCAT)*, 33(4):327–336, diciembre 2008.

[320] A. M. Reina, M. Toro, J. Torres. Integrando Aspectos en MWACSL. En *Actas de los Talleres de las Jornadas de Ingeniería del Software y Bases de Datos (TJISBD)*, volumen 1. Sistedes, setiembre 2007.

[321] A. M. Reina, M. Toro, J. Torres. Generating Domain Specific Aspect Code for Navigation from Platform Specific Models in MWACSL. En *Publicado en las Actas de las XIII Jornadas de Ingeniería del Software y Bases de Datos (JISBD'08)*, páginas 385–390. Sistedes, octubre 2008.

[322] A. M. Reina, J. Torres. Analysing the Navigational Aspect. En *Second Workshop on Aspect-Oriented Software Development of the German Information Society.* Institut für Informatik III, Universität Bonn, febrero 2002. Technical report IAI-TR-2002-1, ISSN 0944-8535.

[323] A. M. Reina, J. Torres. Separating the Navigational Aspect. En *Proceedings of 2nd International Workshop on Aspect Oriented Programming for Distributed Computing Systems (ICDCS 2002), Vol. 2*, julio 2002.

[324] A. M. Reina, J. Torres. Components+Aspects: A General Overview. *Revista Colombiana de Computación*, 5(1):77–95, junio 2004.

[325] A. M. Reina, J. Torres. Implicaciones de Transformaciones Oblicuas en el Desarrollo de un Framework Generador de Aplicaciones Orientadas a Aspectos. *II Taller sobre Desarrollo de Software Dirigido por Modelos, MDA y Aplicaciones (DSDM'05) published as CEUR Workshop Proceedings*, 157, setiembre 2005.

[326] A. M. Reina, J. Torres. Weaving AspectJ by means of Transformations. En *Proceedings of the First Workshop on Models and Aspects - Handling Crosscutting Concerns in MDSD. Workshop held at 19th European Conference on Object-Oriented Programming (ECOOP 2005)*, julio 2005.

[327] A. M. Reina, J. Torres. Using Aspect-Oriented Techniques to Improve the Reuse of Metamodels. *Proceedings of the Second International Workshop on Aspect-Based and Model-Based Separation of Concerns in Software Systems (ABMB 2006)*, 163(2):29–43, abril 2007.

[328] A. M. Reina, J. Torres. Metamodelling and Transforming Sitemaps for Reconciling Information Architecture and Navigation Design. En *Actas de los Talleres de las Jornadas de Ingeniería del Software y Bases de Datos (TJISBD)*, volumen 3. Sistedes, setiembre 2009.

[329] A. M. Reina, J. Torres. Sitemaps From a Model Driven Perspective. A First Step for Bridging the Gap Between Information Architecture and Navigation Design. En *WEBIST 2010. Proceedings of 6th International Conference on Web Information Systems and Technology*, volumen 2, páginas 111–117. INSTICC. Institute for Systems and Technologies of Information, Control and Communication, abril 2010. ISBN: 978-989-674-025-2.

[330] A. M. Reina, J. Torres, M. J. Escalona, J. A. Ortega. Revisiting Requirements in Web Modelling Languages. En *Proceedings of the IADIS International Conference WWW/Internet*, volumen 2, páginas 1065–1067. IADIS Press, noviembre 2003.

[331] A. M. Reina, J. Torres, M. Toro. Aspect-Oriented Web Development vs. Non Aspect-Oriented Web Development. En *AAOS 2003:Analysis of Aspect-Oriented Software (ECOOP 2003)*, julio 2003.

[332] A. M. Reina, J. Torres, M. Toro. Separating Concerns by means of UML-profiles and Metamodels in PIMs. En O. Aldawud y otros [9].

[333] A. M. Reina, J. Torres, M. Toro. Hacia Lenguajes de Metamodelado Orientados a Aspectos. En *Proceedings of the Workshop of Aspect-Oriented Software Development (DSOA'06). Collocated to XI Jornadas de Ingeniería del Software y Bases de Datos (JISBD'06).*, páginas 19–26. Univ. Extremadura, octubre 2006.

[334] A. M. Reina, J. Torres, M. Toro. El Metamodelado de un Framework: Spring Web Flow. En *Actas de los Talleres de las Jornadas de Ingeniería del Software y Bases de Datos (TJISBD)*, volumen 1. Sistedes, setiembre 2007.

[335] A. M. Reina, J. Torres, M. Toro. Improving the Adaptation of Web Applications to Different Versions of Software with MDA. En *Proceedings of the Workshop*

on Adaptation and Evolution in Web Systems Engineering (AEWSE07). Published as: 7th International Conference on Web Engineering. Workshop Proceedings, páginas 101–107, julio 2007.

[336] A. M. Reina, J. Torres, M. Toro. De Flujos de Navegación a Spring Web Flow. Un Primer Acercamiento a las Transformaciones Verticales en MWACSL. En *Actas de los Talleres de las Jornadas de Ingeniería del Software y Bases de Datos (TJISBD)*, volumen 4, páginas 19–28. Sistedes, setiembre 2010.

[337] A. M. Reina, J. Torres, M. Toro, M. J. Escalona. Modelando Aspectos con Lenguajes Específicos de Dominio. En *Taller de Desarrollo de Software Orientado a Aspectos (DSOA'04)*, noviembre 2004.

[338] A. M. Reina, J. Torres, M. Toro, J. A. Álvarez. Concerns vs. Components for Web Development. En *Proceedings of the IADIS International Conference WWW/Internet*, volumen 2, páginas 873–876. IADIS Press, noviembre 2003.

[339] A. M. Reina, J. Torres, M. Toro, J. A. Álvarez. Separación de conceptos y MDA: Arquitectura de un framework. En *I Taller sobre Desarrollo de Software Dirigido por Modelos, MDA y Aplicaciones (DSDM'04)*, noviembre 2004.

[340] A. M. Reina, J. Torres, M. Toro, J. A. Álvarez, J. M. Nieto. Una Experiencia Práctica Reutilizando Aspectos. En *Actas del Taller sobre Desarrollo de Software Orientado a Aspectos (DSOA'03). Taller celebrado en conjunción con las VIII Jornadas de Ingeniería del Software y Bases de Datos*, páginas 3–10, noviembre 2003.

[341] T. Reiter, E. Kapsammer, W. Retschitzegger, W. Schwinger. Model Integration Through Mega Operations. En *Proceedings Of The Workshop On Model-Driven Web Engineering (MDWE 2005)*, julio 2005.

[342] H. Reza, K. Ogaard, A. Malge. A Model Based Testing Technique To Test Web Applications Using Statecharts. En *Proceedings of the Fifth International Conference on Information Technology: New Generations (ICIT-NG'08)*, páginas 183–188. IEEE, 2008.

[343] G. Rossi, A. Nieto, L. Mengoni, L. Nuño Silva. Designing Volatile Functionality in E-Commerce Web Applications. En *Proceedings of EC-Web 2006*, páginas 92–101. Springer-Verlag Lecture Notes in Computer Science, 2006.

[344] J. Rubin, M. Chechik, S. Easterbrook. Declarative Approach for Model Composition. En *International Conference on Software Engineering. Proceedings of the 2008 international workshop on Models in software engineering (MISE 2008)*, páginas 7–13. ACM, 2008.

[345] J. Sánchez Cuadrado, J. García Molina, M. Menárguez Tortosa. RubyTL: A Practical, Extensible Transformation Language. En *Proceedings of the Second European Conference on Model Driven Architecture - Foundations and Applications*, páginas 158–172. Springer-Verlag Lecture Notes in Computer Science, 2006.

[346] S. Sarkar, C. Cleveland. Code Generation Using XML Based Document Transformation, noviembre 2001.

[347] A. Schauerhuber. *Applying Aspect-Orientation to the Model-Driven Development of Ubiquitous Web Applications*. PhD thesis, Faculty of Informatics. Vienna University of Technology, noviembre 2007.

[348] A. Schauerhuber, W. Schwinger, E. Kapsammer, W. Retschitzegger, M. Wimmer, G. Kappel. A Survey on Aspect-Oriented Modeling Approaches. Informe técnico, Vienna University of Technology, abril 2007.

[349] A. Schauerhuber, M. Wimmer, E. Kapsammer. Bridging existing Web Modeling Languages to Model-Driven Engineering: A Metamodel for WebML. En *Proceedings of 2nd International Workshop on Model-Driven Web Engineering (MDWE'06)*, julio 2006.

[350] A. Schauerhuber, M. Wimmer, E. Kapsammer, W. Schwinger, W. Retschitzegger. Bridging WebML to Model-Driven Engineering: From Document Type Definitions to Meta Object Facility. *IET Software*, 1(3):81–97, 2007.

[351] A. H. Schmid, O. Herfort. A Behavioral Semantics of OOHDM Core Features and of Its Business Process Extension. En *Proceedings of International Conference on Web Engineering (ICWE'04)*, volumen 3140, páginas 74–87. Springer-Verlag Lecture Notes in Computer Science, julio 2004.

[352] H. A. Schmid, O. Donnerhak. OOHDMDA. An MDA Approach for OOHDM. En *Proceedings of International Conference on Web Engineering (ICWE'05)*, número 3579, páginas 569–574. Springer-Verlag Lecture Notes in Computer Science, 2005.

[353] D. C. Schmidt. Model-Driven Engineering. *IEEE Computer*, 39(2):25–31, febrero 2006.

[354] D. Schwabe, G. Rossi. Developing Hypermedia Applications using OOHDM. En *Proceedings of Workshop on Hypermedia Development Processes, Methods and Models, Hypertext'98*, 1998.

[355] R. L. Schwartz, T. Phoenix, B. D. Foy. *Learning Perl*. O'Reilly Associates, 4 edición, 2005.

[356] W. Schwinger, N. Koch. *Web Engineering - The Discipline of Systematic Development of Web Applications*, capítulo 3. Modelling Web Applications, páginas 39–64. John Wiley and Sons, 2006.

[357] S. Sendall, W. Kozaczynski. Model Transformation: The Heart and Soul of Model-Driven Software Development. *IEEE Software*, 20(5):42–45, 2003.

[358] M. Shonle, K. Lieberherr, A. Shah. XAspects: An extensible System for Domain-Specific Aspect Languages. En *Companion of 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, páginas 28–37. ACM Press, 2003.

[359] P. Soule. *Autonomics Development: A Domain-Specific Aspect Language Approach*. Autonomic Systems. Springer Basel, 2010.

[360] W. Sousan, V. Winter, M. Zand, H. Siy. ERTSAL: A Prototype of a Domain-Specific Aspect Language for Analysis of Embedded Real-Time Systems. En *Proceedings of 2nd Workshop on Domain Specific Aspect Languages*, DSAL'07, New York, NY, USA, 2007. ACM.

[361] D. Spencer. *A Practical Guide to Information Architecture*. Five Simple Steps, 1 edición, 2010.

[362] J. Sprinkle, A. Agrawal, T. Levendovszky, F. Shi, G. Karsai. Domain Model Translation using Graph Transformations. En *Proceedings of the International Conference Engineering of Computer-Based Systems*, páginas 159–168, 2003.

[363] T. Stahl, M. Völter, J. Bettin, A. Haase, S. Helsen. *Model-Driven Software Development: Technology, Engineering, Management*. Wiley, 2006.

[364] D. Stein, S. Hanenberg, R. Unland. An UML-based Aspect-Oriented Design Notation. En *Proceedings of 1st International Conference on Aspect-Oriented Software Development (AOSD 2002)*, páginas 106–112. ACM Press, abril 2002.

[365] D. Stein, S. Hanenberg, R. Unland. Join Point Designation Diagrams: A Graphical Representation of Join Point Selections. *International Journal of Software Engineering and Knowledge Engineering (IJSEKE). Special Issue on Aspect-Oriented Software Design Models*, 16(3):317–346, 2006.

[366] E. J. Stocks. *Sexy Web Design*, capítulo 4. Navigation and Interaction. En K. Steele [367], 1 edición, marzo 2009.

[367] E. J. Stocks. *Sexy Web Design*. SitePoint Pty. Ltd., 1 edición, marzo 2009.

[368] J. Suzuki, Y. Yamamoto. Extending UML with Aspects: Aspect Support in the Design Phase. En *International Workshop on Aspect-Oriented Programming (ECOOP 1999)*, junio 1999.

[369] G. Taentzer, K. Ehrig, E. Guerra, J. de Lara, L. Lengyel, T. Levendovsky, U. Prange, D. Varro, S. Varro-Gyapay. Model Transformation by Graph Transformation: A Comparative Study. En *In Proceedings of the Workshop on Model Transformation in Practice*, octubre 2005.

[370] É. Tanter, J. Noyé. A Versatile Kernel for Multi-Language AOP. En *Generative Programming and Component Engineering, 4th International Conference (GPCE)*, volumen 3676 de *Lecture Notes in Computer Science*, páginas 173–188. Springer, setiembre 2005.

[371] N. A. Tariq, N. Akhter. Comparison of Model Driven Architecture (MDA) Based Tools. Master thesis, Karolinska University Hospital, 2005.

[372] P. Tarr, H. Ossher, W. Harrison, S. M. Sutton Jr. N Degrees of Separation: Multi-Dimensional Separation of Concerns. En *Proceedings of 21st International Conference on Software Engineering (ICSE 1999)*, páginas 107 – 119. IEEE Computer Society Press, mayo 1999.

[373] AGG Development Team. AGG Homepage. `http://tfs.cs.tu-berlin.de/agg`.

[374] JBoss Community team. Hibernate Website. website:`http://www.hibernate.org/`, January 2011.

[375] MOFLON Development Team. MOFLON. `http://www.moflon.org/`.

[376] RubyTL Development Team. RubyTL. `http://rubytl.rubyforge.org/`.

[377] TCS Development Team. TCS Home Page. `http://www.eclipse.org/gmt/tcs/`.

[378] VMTS Development Team. VMTS Home Page. `http://avalon.aut.bme.hu/~tihamer/research/vmts`.

[379] Velocity Development Team. Velocity Wiki. `http://wiki.apache.org/velocity/FrontPage`, 2007.

[380] S. Tiwari. Implementing Navigation and Pageflows with Seam. *Javalobby*, mayo 2008.

[381] M. Tkatchenko, G. Kiczales. Uniform Support for Modeling Crosscutting Structure. En *Model Driven Engineering Languages and Systems*, volumen 3713 de *Lecture Notes in Computer Science*, páginas 508–521. Springer, 2005.

[382] J. P. Tolvanen, S. Kelly. Defining Domain-Specific Modeling Languages to Automate Product Derivation: Collected Experiences. En *SPLC*, número 3714, páginas 198–209. Springer-Verlag Lecture Notes in Computer Science, octubre 2005.

[383] JET Development Tool. JET, Model To Text Eclipse Tool. `http://www.eclipse.org/modeling/m2t/?project=jet#jet`, 2006.

[384] C. R. Turner, A. Fuggetta, L. Lavazza, A. L. Wolf. Feature Engineering. En *Proceedings of 9th International Workshop on Software Specification and Design*, páginas 162–164, abril 1998.

[385] UWE Project. The UWE Project Web Page:. `http://www.pst.ifi.lmu.de/projekte/uwe/`, 2006.

[386] P. Valderas, V. Pelechano, G. Rossi, S. Gordillo. From Crosscutting Concerns to Web Systems Models. En *Proceedings of the Web Information Systems Engineering (WISE 2007)*, volumen 4831, páginas 573–582. Springer-Verlag Lecture Notes in Computer Science, 2007.

[387] A. Vallecillo, N. Koch, C. Cachero, S. Comai, P. Fraternali, I. Garrigós, J. Gómez, G. Kappel, A. Knapp, M. Matera, S. Meliá, N. Moreno, B. Pröll, T. Reiter, W. Retschitzegger, J. E. Rivera, A. Schauerhuber, W. Schwinger, M. Wimmer, G. Zhang. MDWEnet: A Practical Approach to Achieving Interoperability of Model-Driven Web Engineering Methods. En *In Proceedings of 3rd Model-Driven Web Engineering Workshop (MDWE'07)*, volumen 261. CEUR-WS, 2007.

[388] A. van Deursen, P. Klint, J. Visser. Domain-Specific Languages: An Annotated Bibliography. *ACM SIGPLAN Notices*, 35(6):26–36, 2000.

[389] J. M. Vara, B. Vela, E. Marcos. Transformaciones de Modelos para el Desarrollo de Bases de Datos XML. En *Actas del Taller sobre Desarrollo Dirigido por Modelos. MDA y Aplicaciones.*, volumen 227. CEUR Workshop Proceedings, octubre 2006.

[390] D. Varró, A. Pataricza. Generic and Meta-transformations for Model Transformation Engineering. En *Proceedings of UML 2004: 7th International Conference on the Unified Modeling Language*, volumen 3273, páginas 290–304. Springer-Verlag Lecture Notes in Computer Science, 2004.

[391] E. Vervaet. *The Definitive Guide to Spring Web Flow.* APress, 2008.

[392] E. Visser. *Generative and Transformational Techniques in Software Engineering II*, volumen 5235/2008 de *LNCS*, capítulo WebDSL: A Case Study in Domain-Specific Language Engineering, páginas 291–373. Springer, 2008.

[393] S. Völkel, S. Szchaler, J. Bézivin, D. Kolovos, J. Johannes, J. Reznik, M. Didonet del Fabro, S. Zschaler. MODELPLEX. Workpackage 3: Model Engineering. Deliverable D3.1.a: Model composition. Informe técnico D 3.1a, MODELPLEX Project (Contract n° 034081), octubre 2007.

[394] M. Völter, E. Visser. Language Extension and Composition with Language Workbenches. En *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*, SPLASH'10, páginas 301–304, New York, NY, USA, 2010. ACM.

[395] C. von Flach G. Chavez, C. J. P. de Lucena. Design-level Support for Aspect-Oriented Software Development. En *Workshop on Advanced Separation of Concerns in Object-Oriented Systems (OOPSLA 2001)*, octubre 2001.

[396] W3C. XSL Transformation (XSLT), noviembre 1999.

[397] W3C. XHTML Schema Definition. `http://www.w3.org/2002/08/xhtml/xhtml1-strict.xsd`, Ago 2002.

[398] W3C. OWL Web Ontology Language. `http://www.w3.org/TR/owl-features/`, noviembre 2004.

[399] W3C. XML Path Language (XPath), noviembre 2005.

[400] W3C. XQuery 1.0: An XML Query Language, abril 2005.

[401] W3C. Extensible Markup Language (XML) 1.0 (Fifth Edition). W3C Recommendation 26 November 2008 REC-xml-20081126/, W3C, Nov 2008.

[402] D. Wagelaar, L. Bergmans. Using a Concept-Based Approach to Aspect-Oriented Software Design. En *Workshop on Identifying, Separating and Verifying Concerns in the Design (AOSD 2002)*, marzo 2002.

[403] R. Wagner. *Web Design Before & After Makeovers*, capítulo 3. Navigation Makeovers, páginas 37–58. Wiley Publishing, Inc., 2006.

[404] W. Wang. Evaluation of UML Model Transformation Tools. Diploma thesis, Business Informatics Group. Institut fur Softwaretechnik und Interaktive Systeme, 2005.

[405] WebDSL.org. WebDSL. `http://webdsl.org/`, 2011.

[406] J. Whittle, P. Jayaraman, A. Elkhodary, A. Moreira, J. Araújo. MATA: A Unified Approach for Composing UML Aspect Models Based on Graph Transformation. *Transactions on AOSD*, VI(5560):191–237, 2009.

[407] R.J. Wieringa, J.M.G. Heerkens. Designing Requirements Engineering Research, octubre 2007. Presented at the Workshop on Comparative Evaluation in Requirements Engineering (CERE'07), 15th October 2007, New Delhi.

[408] M. Wimmer, G. Kramler. Bridging Grammarware and Modelware. En *WiSME 2005, 4th Workshop in Software Model Engineering*, octubre 2005.

[409] M. Wimmer, G. Kramler. Pierre-Alain Muller and Michel Hassenforder. En *HUTN as a Bridge between ModelWare and GrammarWare - An Experience Report*, octubre 2005.

[410] M. Wimmer, A. Schauerhuber, W. Schwinger, H. Kargl. On the Integration of Web Modeling Languages: Preliminary Results and Future Challenges. En *Proceedings of 3rd International Workshop on Model-Driven Web Engineering (MDWE'07)*, páginas 255–269, julio 2007.

[411] M. Winckler, P. A. Palanque. StateWebCharts: A Formal Description Technique Dedicated to Navigation Modelling of Web Applications. En *DSV-IS*, volumen 2844 de *Lecture Notes in Computer Science*, páginas 61–76. Springer, 2003.

[412] M. A. Winckler. *StateWebCharts: a Formal Notation for Navigation Modelling of Web Applications*. PhD thesis, Institute de Recherche en Informatique de Toulouse, 2004.

[413] C. Wodtke, A. Govella. *Information Architecture: Blueprints for the Web*. New Riders, 2 edición, 2009.

[414] C. Wodtke, A. Govella. *Information Architecture: Blueprints for the Web*, capítulo 8. The Tao of Navigation. En K. Steele [413], 2 edición, 2009.

[415] XFramer-Team. XFramer. Frame Technology with the Concepts of Frames and Slots. `http://www.xframer.com/`, 2007.

[416] A. A. Zakaria, H. Hosny, A. Zeid. A UML Extension for Modeling Aspect-Oriented Systems. En M. Kandé y otros [207].

[417] G. Zhang. *Aspect-Oriented State Machines*. PhD thesis, Fakultat fur Mathematik, Informatik und Statistik der Ludwig Maximilians Universitat Munchen, 2010.

[418] G. Zhang, H. Baumeister, N. Koch, A. Knapp. Aspect-Oriented Modeling of Access Control in Web Applications. En *6th International Workshop on Aspect-Oriented Modeling*, marzo 2005.

Listado de Acrónimos

AGE: Agile Generative Enviroment, 83
AGG: Attributed Graph Grammar, 138
AI: Arquitectura de la Información, 87, 155
AOA: Análisis Orientado a Aspectos, 47
API: Interfaz de Programación de Aplicaciones, 74
ATL: Atlas Transformation Language, 83

CTL: Computational Tree Logic, 116

DOA: Diseño Orientado a Aspectos, 47
DSAL: Domain-Specific Aspect Language, 42
DSDM: Desarrollo de Software Dirigido por Modelos, 4, 15, 51, 59, 60, 62, 65, 78, 85, 135, 150, 196
DSL: Domain Specific Language, 81, 137, 182
DSL's: Domain Specific Languages, 4
DSOA: Desarrollo de Software Orientado a Aspectos, 4, 15, 24, 36, 38, 44, 87, 143
DTD: Document Type Definition, 135

EJB: Enterprise Java Beans, 68, 76
EMF: Eclipse Modelling Framework, 78–80
EMOF: Essential Meta-Object Facilities, 83
EMP: Eclipse Modeling Project, 190

FARNav: Formal Approach for Rich Navigation, 87, 114
FC: Filtros de Composición, 38

GMT: Generative Modeling Technologies, 190
GPAL: General-Purpose Aspect Language, 42

HBMS: Hyperdocument Model Based on Statecharts, 113
HDM: Hypertext Design Model, 138

HUTN: Human-Usable Textual Notation, 67

IDM: Ingeniería Dirigida por Modelos, 60
IW: Ingeniería Web, 4, 155

JAC: Java Aspect Components framework, 21
JET: Java Emitter Templates, 80
JSF: Java Server Faces, 146
JSP: Java Server Pages, 195

LAED: Lenguaje de Aspectos Específico de Dominio, 42, 44
LAEDs: Lenguajes de Aspectos Específicos de Dominio, 19, 43, 44
LAPG: Lenguaje de Aspectos de Propósito General, 42
LAPGs: Lenguajes de Aspectos de Propósito General, 144
LHS: Left Hand Side, 75

MAA: Modelo de Arquitectura Orientado a Aspectos, 54
MDA: Model Driven Architecture, 4, 32, 51, 59, 65, 68–71, 73, 78, 84, 121, 136–139
MDE: Model-Driven Engineering, 60
MDR: Metadata Repository, 78
MDSD: Model-Driven Software Development, 60
MIC: Model-Integrated Computing, 4, 65, 67
MOA: Modelado Orientado a Aspectos, 20
MOF: Meta Object Framework, 65, 67, 69, 78, 79, 135, 138
MVC: Model View Controller, 137

OMG: Object Management Group, 68, 69, 78–80, 83
OOHDM: Object-Oriented Hypermedia Design Method, 87, 105, 136
OOWS: Object-Oriented Web Solutions, 87, 110
OWL: Ontology Web Language, 108

PA: Programación Adaptable, 38

POA: Programación Orientada a Aspectos, 36, 38, 45
POJO: Plain Old Java Object, 186

QVT: Queries Views Transformations, 69, 76, 81–83

RDF: Resource Description Framework, 108
RHS: Right Hand Side, 75

SAC: Separación Avanzada de Conceptos, 38
SDM: Story Driven Modelling, 83
SEAL: Semantic-based Aspect-Oriented Adaptation Language, 127
SeRQL: Sesame RDF Query Language, 108
SF: Software Factories, 4, 65
SHDM: Semantic Hypermedia Design Method, 137
SMC: Separación Multidimensional de Competencias, 38, 45
SOP: Subject-Oriented Programming, 21
SVM: Symbolic View Modifier, 116
SWF: Spring Web Flow, 178, 182

TCS: Textual Concrete Syntax, 81
TMF: Textual Modelling Framework, 80
Turtle: Terse RDF Triple Language, 108

UID: User Interaction Diagram, 125
UML: Unified Modeling Language, 49, 51, 52, 54–56, 61, 65, 67, 69, 71, 73–76, 78, 80, 136–139
UWE: UML-based Web Engineering, 87, 103

VTL: Velocity Template Language, 80

WAAT: Web Applications Analysis and Testing, 87, 116
WebML: Web Modeling Language, 87, 100, 135
WebSA: Web Software Architecture, 139

XMI: XML Metadata Interchange, 62, 69, 73, 75, 79, 136
XML: Extensible Markup Language, 78, 135

`XSD`: XML Schema Definition, 178

`XSLT`: Extensible Stylesheet Language Transformations, 109, 135