UNIVERSIDAD COMPLUTENSE DE MADRID
FACULTAD DE INFORMÁTICA
Departamento de Ingeniería del Software e Inteligencia Artificial

TESIS DOCTORAL

Una aproximación dirigida por modelos para la caracterización de la capa de presentación web de aplicaciones empresariales

MEMORIA PARA OPTAR AL GRADO DE DOCTOR

PRESENTADA POR

Humberto Cortés Benavides

Director

Antonio Navarro Martín

Madrid, 2018

UNA APROXIMACIÓN DIRIGIDA POR MODELOS PARA LA CARACTERIZACIÓN DE LA CAPA DE PRESENTACIÓN WEB DE APLICACIONES EMPRESARIALES

Humberto Javier Cortés Benavides

Facultad de Informática

Universidad Complutense de Madrid

Director:

Antonio Navarro Martín

abril 2017

UNA APROXIMACIÓN DIRIGIDA POR MODELOS PARA LA CARACTERIZACIÓN DE LA CAPA DE PRESENTACIÓN WEB DE APLICACIONES EMPRESARIALES

Humberto Javier Cortés Benavides

Facultad de Informática

Departamento de Ingeniería del Software e Inteligencia Artificial

Universidad Complutense de Madrid

Memoria para optar al grado de Doctor

Director:

Antonio Navarro Martín

abril 2017

A Tatiana, por ser mi polo a tierra en los momentos de máxima exigencia.

AGRADECIMIENTOS

Este trabajo ha tenido el privilegio de contar con el Doctor Antonio Navarro Martín como su director. Sin sus amplios conocimientos, tanto teóricos como prácticos, unidos a su admirable dedicación y experiencia, este trabajo no podría haber salido a la luz. Muchas gracias Antonio por mostrarme que es posible aplicar los conocimientos teóricos y/o abstractos al desarrollo industrial.

Me gustaría reconocer el ingente trabajo desarrollado por la comunidad de la Ingeniería Web en los últimos años, y agradecer a todos los autores del sector industrial referenciados en este trabajo por su gran aporte a consolidar las buenas prácticas en el desarrollo de software.

CONTENIDO

LISTA DE TABLAS

LISTA DE FIGURAS

ABSTRACT

Nowadays UML is the most successful notation for the design of object-oriented applications. However, plain UML is not enough to characterize the Web presentation tier of enterprise applications. The Web engineering community has provided the *navigation map* concept to deal with the complexity of Web applications. Following this concept we developed NMMp, a design notation that, as the Web engineering notations, provides an abstract vision of the navigation structure of the Web presentation tier, but encouraging the explicit inclusion of multitier, SOA and security design-level patterns in the models, which are widely used in industry.

However, NMMp does not take into account the characterization of specific frameworks extensively used in the development of the Web presentation tier of modern enterprise applications. Generally, these frameworks support the navigational, structural and role-based access control (RBAC) features present in these applications. Following the NMMp philosophy, we have developed the Enterprise Web Application Extension (E-WAE) approach, a lightweight UML extension set for the modeling of these features of the Web presentation tier of enterprise applications.

By means of PIM-level models, E-WAE provides an abstract vision of the Web presentation tier independently of architectural details or programming languages. However, the approach follows a Model-Driven Development (MDD) approach, which enables the manual or automatic generation of intermediate PSM-level models, which characterize the components implemented in JSF and ASP.NET MVC frameworks and allows the inclusion of multitier, SOA and security design-level patterns in the models.

These PSM models provide a semantic that can be directly translated into platform-specific components, avoiding bound code generation and maintenance to any tool. In addition, these semantics make it easier to read the UML models (by thinking in terms of the corresponding platform-specific components) and clarify ambiguous modeling. Furthermore, the present work defines a set of transformation rules that translate the PSM models into code. It thereby defines a complete executable specification of the models, clarifying their translation to platform-specific code and thus closing the gap between modeling and code.

In addition, the generated code can be used as a low-cost mockup for early client validation of the Web presentation tier of enterprise applications, because by means of the presentation structure, they describe the functionalities supported by applications.

These mockups, that are called *navigational mockups*, permit end-users to interact with the application, allowing the whole presentation tier to be validated in the field and listening to their feedback.

Using the automatic transformations the mockups can be generated in conjunction with end-users, applying agile principles to the development process. However, the mockups are generated from well defined PSM UML models that can be enriched by developers with the architectural and design patterns described in the pattern catalogs. Thus, the approach entrusts developers with code development and is not limited by specific tools (*black box wizards*) to generate the code.

Nowadays there is little research or literature about the relationship between well-architected designs and agile approaches. Well-architected design means that enough time has been spent in architectural planning to ensure the effectiveness and maintainability of the solution, as against agile approaches that encourage delivery to end-users as soon as possible. The presented approach can be seen as an attempt to reconcile well-architected design and agile approaches.

RESUMEN

Actualmente, UML es la notación de diseño más ampliamente usada en el diseño y desarrollo de software orientado a objetos. Sin embargo, UML estándar no es suficiente para caracterizar la capa de presentación Web de las aplicaciones empresariales. La comunidad de la Ingeniería Web ha proporcionado el concepto de *mapa navegacional* para tratar de gestionar la complejidad inherente a las aplicaciones Web. Siguiendo este concepto, hemos desarrollado NMMp, una notación de diseño que, como las notaciones de la Ingeniería Web, proporciona una visión abstracta de la estructura navegacional de la capa de presentación Web. Sin embargo, a diferencia de estas, promueve la inclusión explícita de todo el catálogo de patrones arquitectónicos y de diseño en los modelos, los cuales, son ampliamente usandos en el sector industrial.

Sin embargo, NMMp no tiene en cuenta la caracterización de frameworks específicos usados ampliamente en el desarrollo de la capa de presentación Web de aplicaciones empresariales modernas. Generalmente, estos frameworks soportan el desarrollo de la navegación, la estructura de presentación y el control de acceso basado en roles (RBAC), características presentes en la mayoría de este tipo de aplicaciones. Siguiendo la filosofía de NMMp, hemos desarrollado el enfoque Enterprise Web Application Extensión (E-WAE), como un conjunto de extensiones UML que soportan el modelado de estas características presentes en frameworks específicos.

Por medio de modelos PIM, E-WAE proporciona una visión abstracta de la capa de presentación Web, independientemente de detalles arquitectónicos y conceptos de programación. Sin embargo, el enfoque sigue la filosofía de desarrollo dirigida por modelos (MDD) para generar, manual o automáticamente, modelos PSM intermedios. Estos modelos caracterizan componentes implementados en los frameworks JSF y ASP.NET MVC, y permiten la inclusión explicita de todo el catálogo de patrones arquitectónicos y de diseño en los modelos.

La semántica de los modelos PSM esta dada por los componentes presentes en los frameworks, lo cual permite su transformación directa a código, evitando vincular la generación y mantenimiento de este con herramientas específicas. Además, esta semántica facilita la lectura de los modelos (pensando en términos de los correspondientes componentes implementados en los frameworks) y evita la ambigüedad en el modelado. Así mismo, el trabajo presentado define un conjunto de reglas para transformar los

modelos PSM en código. Por tanto, se proporciona una especificación completa y ejecutable de los modelos cerrando la brecha entre modelos y código.

Por otro lado, el código generado puede ser usado como mockups para validar la capa de presentación Web en una etapa temprana de desarrollo. Por medio de la capa de presentación, estos mockups, que hemos llamado *mockups navegacionales*, describen las funcionalidades soportadas por las aplicaciones, permitiendo a los usuarios finales interactuar con las aplicaciones sin necesidad de ejecutar lógica de negocio, validar la capa de presentación Web en el campo y escuchar sus opiniones.

Por medio de la ejecución automática de las reglas de transformación, los mockups pueden ser generados en conjunto con los usuarios finales, obteniendo los beneficios de los procesos ágiles en el proceso de desarrollo. Sin embargo, los mockups son generados desde modelos UML bien definidos, que pueden ser extendidos por los desarrolladores con patrones arqutectónicos y de diseño. Por tanto, el enfoque confía el desarrollo a los programadores y no está limitado por herramientas específicas (*black-box wizards*) para generar el código.

Actualmente, no hay mucha investigación y literatura acerca de la relación entre el diseño arquitectónico y los procesos ágiles. El diseño arquitectónico implica invertir el tiempo suficiente en el diseño del sistema para asegurar la efectividad y mantenibilidad de la solución, lo cuál está en conflicto con los procesos ágiles que promueven las entregas rápidas para mantener la alta interacción con los usuarios finales. Desde este punto de vista, el enfoque presentado puede ser visto como un intento de reconciliar el diseño de la arquitectura y los procesos ágiles.

1 INTRODUCCIÓN

1.1 Contexto de la investigación

El diseño, desarrollo y mantenimiento de las aplicaciones empresariales es una labor compleja. Fowler (2002) define este tipo de aplicaciones por medio de las siguientes características: (i) administran grandes volúmenes de datos persistentes; (ii) los datos persistentes son accedidos y manipulados concurrentemente; (iii) están compuestas por una gran variedad y cantidad de interfaces de usuario; (iv) tienen necesidades de integración con otras aplicaciones empresariales; (v) implementan reglas de negocio complejas; y (vi) tienen que ser mantenidas en el tiempo.

La adecuada definición de la arquitectura software, junto con el desarrollo de modelos apropiados que la representen, son aspectos claves para el desarrollo y mantenimiento de este tipo de aplicaciones. En la industria, la arquitectura multicapa (*multitier architecture*) ha demostrado ser una de las arquitecturas más efectivas para el desarrollo de aplicaciones empresariales (Alur et al., 2003; Crawford & Kaplan 2003; Fowler, 2002). Siguiendo esta arquitectura, las aplicaciones empresariales se dividen en un mínimo de tres capas que pueden ser desarrolladas, mantenidas y desplegadas independientemente: capa de presentación, capa de negocio y capa de integración.

La capa de presentación es responsable de implementar la funcionalidad requerida para que los usuarios interactúen con la aplicación. La capa de negocio implementa las funcionalidades de la aplicación. Esta capa define los componentes que ejecutan los procesos de negocio de acuerdo a sus reglas, así como los componentes que informan sobre la situación de estos procesos. La capa de integración comprende la implementación del acceso a datos y las funcionalidades transversales que pueden ser usadas desde

diferentes capas como seguridad en el acceso a datos, monitorización, configuración, etc. Por encima de la capa de presentación está la capa de cliente (actores que acceden a las funciones de la aplicación), y por debajo de la capa de integración está la capa de recursos (sistemas de gestión de bases de datos u otras aplicaciones que contienen a las bases de datos).

Respecto al modelado, UML es, con mucha diferencia, el lenguaje más ampliamente usado, tanto en la academia como en la industria. Todas las recomendaciones curriculares de instituciones como ACM e IEEE, La Guía al Cuerpo de Conocimiento de la Ingeniería de Software (SWEBOK Guide) (Bourque, P. & Fairley, P., 2014) y los principales catálogos de patrones arquitectónicos y de diseño (Alur et al., 2003; Crawford & Kaplan 2003, Fowler, 2002) coinciden en utilizar UML como lenguaje de modelado en el desarrollo de aplicaciones empresariales y software orientado a objetos en general (de hecho, la mayoría de patrones arquitectónicos y de diseño se definen usando UML).

Sin embargo, la mayoría de aplicaciones empresariales proveen una capa de presentación Web que UML standard no es capaz de modelar. Entre otros elementos, esta capa de presentación incluye páginas y enlaces (*links*), los cuales no se pueden caracterizar con conceptos del diseño orientado a objetos (Conallen, 1999; Cortés & Navarro, 2014). Para tratar de gestionar la complejidad inherente a la capa de presentación Web, la comunidad de la Ingeniería Web ha aportado del concepto de *mapa navegacional*. Los mapas navegacionales son una herramienta muy efectiva para caracterizar, entre otros elementos, páginas y sus relaciones, describiendo, desde un punto de vista global, los principales caminos a través de la interfaz de usuario (Abrahao et al., 2003; Conallen, 1999; Cortés & Navarro, 2014; Han & Hofmeister, 2005).

Actualmente podemos encontrar un número significativo de enfoques y/o herramientas que soportan el diseño y desarrollo de aplicaciones Web. Unos provienen de la academia, que en este trabajo hemos llamado *enfoques orientados al modelado* y otros, más heterogéneos, provienen de la industria que hemos llamado *enfoques orientado al código*. Algunos de estos enfoques proporcionan herramientas visuales para la generación de aplicaciones Web completas y listas para ser ejecutadas, y otros promueven el uso de patrones arquitectónicos y de diseño para generar *parte* de una aplicación Web. En cualquier caso, ninguno de los enfoques orientados al código proporciona una infraestructura de ingeniería de software dedicada al diseño y mantenimiento de las aplicaciones.

Los enfoques orientados al modelado proveen notaciones bien definidas que se han constituido en verdaderas metodologías de ingeniería Web. Sin embargo, en nuestra opinión, estas metodologías comparten dos importantes características: (i) no están basadas en UML, o aunque definan perfiles UML, estos son tan complejos que en la práctica son nuevas notaciones; y (ii) están limitadas por la expresividad de los elementos de modelado que definen, propios del dominio de la aplicación y lejanos del código final.

En este contexto, la complejidad de los perfiles UML se refleja en el alto número de estereotipos que definen y su naturaleza abstracta, caracterizando conceptos del dominio en lugar de conceptos relacionados al código. Estas metodologías no promueven representaciones UML de código con clases directamente traducibles a código orientado a objetos, sino representaciones propietarias de aplicaciones Web. Aunque en algunos casos los modelos pueden ser desarrollados con herramientas UML CASE genéricas, la falta de definición de modelos PSM que les den un significado en términos de código a los estereotipos que definen, impide obtener código directamente a partir de ellos. Se requiere por tanto de herramientas CASE propietarias para obtener el código desde estas representaciones abstractas de aplicaciones Web. Estas herramientas actúan como generadores propietarios de código (*black-box wizards*) que vinculan el mantenimiento de las aplicaciones necesariamente a su uso.

Además, dado que la semántica de los elementos de modelado que definen no se describe en términos de código, es muy difícil, si no imposible, reutilizarlos en modelos arbitrarios que implementen patrones arquitectónicos y de diseño. Por tanto, cuando se utilizan estas metodologías, las aplicaciones están limitadas a los patrones de diseño predefinidos en la herramienta de generación de código.

Por otro lado, la validación de requisitos es otro de los grandes desafíos de las aplicaciones empresariales. Desde los procesos de desarrollo de software tradicionales, como el incremental, hasta los ágiles, como programación extrema o Scrum, apoyan el uso de mockups como herramientas para la validación de requisitos (Sommerville, 2010). Los mockups se han convertido en un excelente punto de partida para el desarrollo de software, ya que por medio de la estructura de la interfaz de usuario describen todas las funcionalidades soportadas por la aplicación. Si los mockups son interactivos, es decir, soportan la interacción de los usuarios finales, permiten además validar la capa de presentación en el campo, observar a los usuarios finales en acción, analizar su comportamiento en el tiempo y escuchar sus opiniones. Los beneficios de esta clase de

mockups ha sido reconocida tanto en la academia como en la industria (Ricca, F. et al., 2010; Esposito, 2016).

Sin embargo, por lo general, los mockups interactivos se utilizan para refinar los requisitos por medio de la demostración de características específicas a los usuarios. Estos mockups son rápidamente desarrollados sin tener en cuenta los requisitos no funcionales como rendimiento, seguridad, robustez, disponibilidad o usabilidad, y por tanto no con la idea de ser reutilizados en el proceso de desarrollo de las aplicaciones finales. Los desarrolladores desechan los mockups y empiezan el proceso de desarrollo implementado los requisitos identificados (Rivero et al., 2010).

No hay mucha investigación y literatura acerca de la aplicación de los procesos ágiles en el desarrollo de aplicaciones empresariales (Breivold et al., 2010; Buschmann, F., & Henney, K., 2013). El diseño de la arquitectura implica invertir esfuerzo y tiempo suficiente para definir qué componentes forman el sistema, cómo se relacionan entre ellos y cómo mediante su interacción cumplen con los requisitos funcionales especificados, cumpliendo además con los requisitos no funcionales. Sin embargo, este diseño de arquitectura y su posterior mantenimiento causa un conflicto con los procesos ágiles que promueven las iteraciones de desarrollo rápidas para mantener la alta interacción con los usuarios.

1.2 Objetivo de la tesis

El presente trabajo propone un enfoque para soportar el diseño, desarrollo y mantenimiento de la capa de presentación Web de aplicaciones empresariales modernas siguiendo los dos principios de la ingeniería del software mencionados: la arquitectura y el modelado del software. Respecto a la arquitectura, el enfoque promueve el uso de la arquitectura multicapa incluyendo sus patrones de diseño asociados. Respecto al modelado, parte de UML y lo extiende para usarlo como lenguaje de modelado de capas de presentación Web implementadas con frameworks empresariales, reutilizando todo el conocimiento, experiencia y herramientas asociadas al mismo.

Por otro lado, el enfoque presentado tiene como objetivo soportar la validación de requisitos en las aplicaciones empresariales modernas a través de *mockups navegacionales*. Los mockups navegacionales son implementaciones completamente funcionales de la capa de presentación Web que ofrece una perspectiva real de la aplicación en desarrollo, sin necesidad de ejecutar lógica de negocio. Como los procesos ágiles, los mockups navegacionales se implementan a través de ciclos de desarrollo

cortos, pero como los procesos tradicionales están soportados por modelos UML bien definidos. Por tanto, el enfoque propuesto también puede ser visto como un intento de reconciliar los procesos ágiles con el diseño de la arquitectura.

1.3 Desarrollo de la tesis

Este trabajo ha tomado como base la notación de diseño NMM (*Navigation Maps Modeling*) (Navarro et al., 2008) desarrollada en la academia para caracterizar de forma abstracta mapas navegacionales con independencia de detalles arquitectónicos y conceptos de programación. NMM tiene un desarrollo formal que permiten transformar de forma explicita los modelos NMM en modelos UML-WAE (*Web Application Extension*) (Conallen, 1999). Sin embargo, la falta de integración de la notación NMM con herramientas UML CASE genéricas es un serio obstáculo en este proceso de transformación y su aplicación en el desarrollo industrial.

Para salvar este obstáculo, en este trabajo se ha desarrollado NMMp (NMM profile) (Cortés & Navarro, 2014), una extensión UML basada en NMM implementada utilizando el concepto de perfiles UML (OMG, 2010). NMMp hace posible explotar todas las características de NMM en herramientas UML CASE genéricas, incluyendo la capacidad de transformar automáticamente los modelos NMM en modelos UML-WAE a través de la ejecución de un conjunto de reglas de transformación basadas en el estándar QVT (*Query/View/Transformation*) (OMG, 2008). Además, con NMMp los modelos UML-WAE obtenidos pueden ser directamente integrados con el diseño UML del resto de componentes de una aplicación Web dentro de la herramienta UML CASE seleccionada.

Sin embargo, en la práctica, dado que NMMp define la estructura navegacional en términos de transiciones de un página a otra por medio de anclas de recuperación (*retrieval anchors*) y anclas computacionales (*computing anchors*), que dan acceso a páginas estáticas y dinámicas respectivamente, es adecuado para modelar aplicaciones Web en las cuales no es necesario el uso de frameworks empresariales para la implementación de la capa de presentación. Las anclas de recuperación y computacionales ocultan componentes del lado del servidor involucrados en el enrutamiento de páginas, componentes que ejecutan procesos computacionales y los datos que fluyen entre las páginas y estos procesos. Estos componentes son generados de forma genérica en el proceso de transformación automático a modelos UML-WAE. Por tanto, estos modelos siguen siendo diseños abstractos que están aún lejos del nivel de implementación.

Así, de acuerdo al enfoque de la arquitectura dirigida por modelos (*Model-Driven Architecture*, MDA) (OMG, 2003), los modelos NMMp son Modelos Independientes de la Plataforma (*Platform Independent Models*, PIM) que omiten detalles arquitectónicos, y los modelos UML-WAE generados son PIMs que modelan arquitecturas concretas pero no caracterizan componentes de plataformas específicas (*Platform Specific Models*, PSM). Por tanto, los modelos NMMp son muy útiles para modelar aplicaciones Web con arquitecturas concretas, pero la falta de soporte a frameworks industriales dificulta su aplicación en el desarrollo de aplicaciones empresariales modernas.

Reorientando la filosofía de NMMp/UML-WAE al desarrollo de aplicaciones empresariales modernas, en este trabajo hemos definido dos extensiones UML para permitir modelar mapas navegacionales usando componentes Web implementados en dos frameworks empresariales ampliamente utilizados en la industria: JSF y ASP.NET MVC. Los mapas navegacionales desarrollados con estas dos extensiones, que hemos llamado WAE4JSF y WAE4.NET (abreviados, WAE4x) (Cortés & Navarro, 2016) respectivamente, son modelos PSM con una semántica dada por los componentes presentes en los frameworks que permite la transformación directa a código asistiendo el proceso de desarrollo de la capa de presentación Web de aplicaciones empresariales modernas.

La semántica de los perfiles WAE4x permite una transformación directa de los elementos presentes en los modelos a componentes específicos de la plataforma, evitando acoplar la generación y mantenimiento de código con herramientas específicas. De hecho, el presente trabajo implementa un conjunto de reglas automáticas para transformar los modelos WAE4x en código. Por tanto se define una especificación completa y ejecutable de los modelos cerrando así la brecha entre modelos y código.

Sin embargo, dado que la experiencia con NMMp demuestra que el uso de modelos PIM es muy valioso en el proceso de desarrollo porque permiten expresar mapas navegacionales independientemente de conceptos de programación, en este trabajo hemos definido *Enterprise WAE* (E-WAE), una extensión UML que añade un nivel de abstracción sobre los perfiles WAE4x. De esta forma, E-WAE también facilita modelar las características comunes implementadas en los frameworks empresariales. Para demostrar esta capacidad, E-WAE define elementos que representan plantillas y sus regiones, y elementos de control de acceso basado en roles (RBAC), dos característica comunes implementadas en los frameworks JSF y ASP.NET MVC.

Por tanto, E-WAE permite caracterizar mapas navegacionales utilizando componentes propios de aplicaciones empresariales modernas, como ejecución de procesos computacionales, objetos de transferencia de datos, componentes que representan plantillas y sus regiones, y elementos de control de acceso basado en roles (RBAC) entre otros. Los mapas navegacionales desarrollados con E-WAE son modelos PIM que facilitan la comunicación entre desarrolladores y usuarios finales, y actúan como modelos fuente que facilitan la portabilidad y reusabilidad, dos de los atributos fundamentales del software de acuerdo a ISO/IEC 9126 (Jung et al., 2004).

De esta forma, los modelos E-WAE pueden ser vistos como diseños de alto nivel de una aplicación, mientras los modelos WAE4x pueden ser vistos como diseños detallados de la misma aplicación. Además, el hecho de que los perfiles UML compartan un núcleo semántico común (OMG, 2013), facilita la definición de reglas de transformación entre modelos E-WAE y modelos WAE4x. Con este concepto en mente, hemos definido un conjunto de reglas de transformación explícitas para generar modelos PSM WAE4x a partir de modelos PIM E-WAE. Estas reglas de transformación pueden ser aplicadas manualmente, prescindiendo de herramientas específicas, o automáticamente, a través de la ejecución de las transformaciones QVT que hemos implementado con base en la descripción de las reglas manuales, de acuerdo al standard OMG.

Dado que los perfiles E-WAE y WAE4x pueden ser desplegados en cualquier herramienta UML CASE, y las transformaciones PIM a PSM y PSM a código se definen de forma explícita, se puede afimar que E-WAE promueve su implementación automática, permitiendo de esta forma que la generación y mantenimiento de código no dependa de herramientas específicas (*black-box wizards*). Además, dado que los frameworks JSF y ASP.NET MVC han sido diseñados para desarrollar la capa de presentación Web de aplicaciones empresariales, si se utiliza los modelos E-WAE y WAE4x para modelar la capa de presentación y UML plano para modelar el resto de capas, es posible modelar una aplicación empresarial completa reutilizando los patrones arquitectónicos y de diseño suficientemente probados en la industria.

Resultados alcanzadosComo los enfoques orientados al modelado, E-WAE es una notación bien definida para modelar la capa de presentación Web de aplicaciones empresariales modernas. Sin embargo, E-WAE y WAE4x promueven el desarrollo siguiendo la filosofía UML: el número de nuevos estereotipos es mínimo para modelar el dominio específico, existe una definición semántica de los estereotipos en términos de código y los modelos están basados en diagramas UML de clase y de secuencia, como

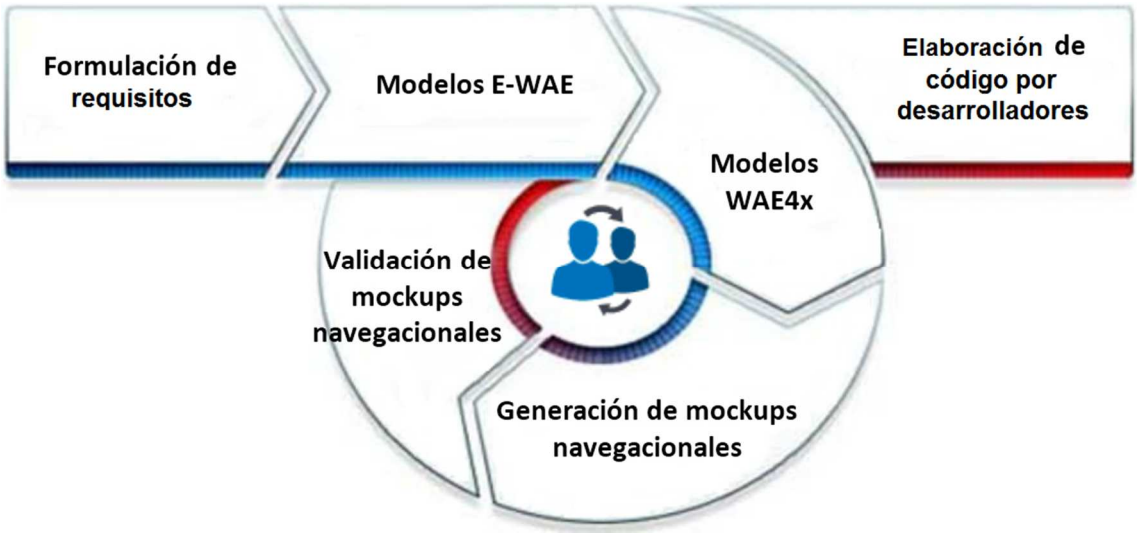
determina el Proceso Unificado Racional (RUP) (Jacobson et al., 1999). Por tanto, los modelos desarrollados con el enfoque presentado no dependen de herramientas propietarias, son legibles para los programadores delegando en ellos el desarrollo del software, y permiten la libre inclusión de patrones arquitectónicos y de diseño, como el desarrollo basado en el proceso unificado promueve. En consecuencia, el enfoque propuesto puede ser visto como un enfoque mixto que toma las ventajas de los enfoques orientados al modelado y del proceso unificado de desarrollo que promueve el uso de patrones arquitectónicos y de diseño.

Como se mencionó anteriormente, el código obtenido desde los modelos WAE4x, es un código listo para ser ejecutado en su respectivo IDE (Eclipse para WAE4JSF y Microsoft Visual Studio para WAE4.NET), en este sentido, el enfoque presentado genera automáticamente mockups interactivos que pueden ser utilizados para validar la capa de presentación Web en una etapa temprana de desarrollo y aisladamente de la ejecución de lógica de negocio.

Para obtener los beneficios de los procesos ágiles con E-WAE, se requiere implementar las reglas de transformación bien definidas en herramientas que permita su ejecución automática. El presente trabajo ha implementado las reglas de transformación PIM2PSM conforme al estándar QVT (*Query/View/Transformation*) (OMG, 2008) soportado por la herramienta CASE Borland Together y las reglas de transformación PSM2Code a través de un conjunto de reglas XSLT compatibles sólo con Borland Together.

De esta forma, como se ilustra en la Figura 1.1, los mockups generados, al igual que en los procesos ágiles, pueden ser desarrollados a través de ciclos iterativos cortos involucrando a los usuarios finales en cada iteración para descubrir o refinar requisitos funcionales. Los desarrolladores definen los mapas navegacionales y los usuarios validan los mockups interactivos generados a partir de ellos. Pero además, dado que los mapas navegacionales, como en los procesos tradicionales, son modelos UML, pueden ser utilizados como documentación bien definida que asegura la calidad y legibilidad suficiente para ser extendidos por los desarrolladores con otros modelos que incluyan patrones arquitectónicos y de diseño, generando, por tanto, aplicaciones completas que implementen una arquitectura definida por los desarrolladores.

Figura 1.1: Flujo de información propuesto por E-WAE y WAE4x



1.4 Publicaciones

Durante el desarrollo de este trabajo, autor y director han publicado los resultados del mismo en las siguientes revistas y foros:

- Navarro, A., & Cortés, H. (2011). Metamodeling or profiling: a practical case in the web engineering domain. In *XVII Congreso Argentino de Ciencias de la Computación.* Artículo que recoje las lecciones aprendidas sobre meta-modelado y desarrollo de perfiles UML que sirvieron de base para el desarrollo de la notación NMMp.
- Cortés, H., & Navarro, A. (2014). NMMp: A Model-Driven UML Extension for the Description of Navigation Maps for Web Applications. *International Journal of Software Engineering and Knowledge Engineering*, 24(03), 391-417. doi:10.1142/S0218194014500156. Factor de impacto ISI JCR Computer Science, Software Engineering: 0,240. Posición: 101/106. Artículo que recoje el desarrollo de la notación NMMp.
- Cortés, H., & Navarro, A. (2016). MDD Inclusion of Navigational, Structural and RBAC Elements for JSF and ASP.NET MVC Frameworks in UML Models. *Fourth International Conference on Technological Ecosystems for Enhancing Multiculturality*, Salamanca, Spain, November 02 - 04, 2016. ACM Press. Artículo que recoje el desarrollo de los perfiles WAE4JSF y WAE4.NET.

1.5 Organización de la memoria

Este trabajo se divide en distintos capítulos. El Capítulo 2 revisa el estado de arte, contextualizando el trabajo presentado en esta memoria y aclarando sus aportes. El Capítulo 3 presenta el desarrollo de NMMp incluyendo las transformaciones a UML-WAE. El Capítulo 4 presenta el desarrollo de los perfiles WAE4x y su traducción a código ejecutable. El Capítulo 5 presenta el desarrollo de E-WAE y su filosofía de desarrollo a través de tres ejemplos basados en aplicaciones reales. Finalmente el Capítulo 6 presenta las conclusiones y trabajo futuro. Así mismo se incluyen cuatro anexos que ilustran, a manera de ejemplo, el código de las transformaciones automáticas de los modelos E-WAE a WAE4x, y de estos a código. No se han incluido en esta memoria todas las transformaciones debido a la extensión de las mismas.

2 ESTADO DEL ARTE

2.1 Introducción

El trabajo relacionado en el área del modelado de la ingeniería web es muy extenso, y por si sólo bien merece un libro (Rossi et al., 2007). En esta sección se analizan las metodologías más relevantes en relación al trabajo propuesto en esta memoria. A continuación se analizarán diversas aproximaciones relacionadas para finalizar haciendo una comparativa de las mismas, entre sí, y en relación al trabajo presentado en esta memoria.

2.2 Web Modeling Language - WebML

WebML (Ceri et al., 2002) es un lenguaje de modelado conceptual de alto nivel concebido para el desarrollo de aplicaciones Web que hacen uso intensivo de datos (*data-intensive Web applications*). Estas se definen como “aplicaciones cuyo propósito principal es dar acceso a contenido bien estructurado y las cuales son predominantes en términos de volumen y valor comercial en el mercado”.

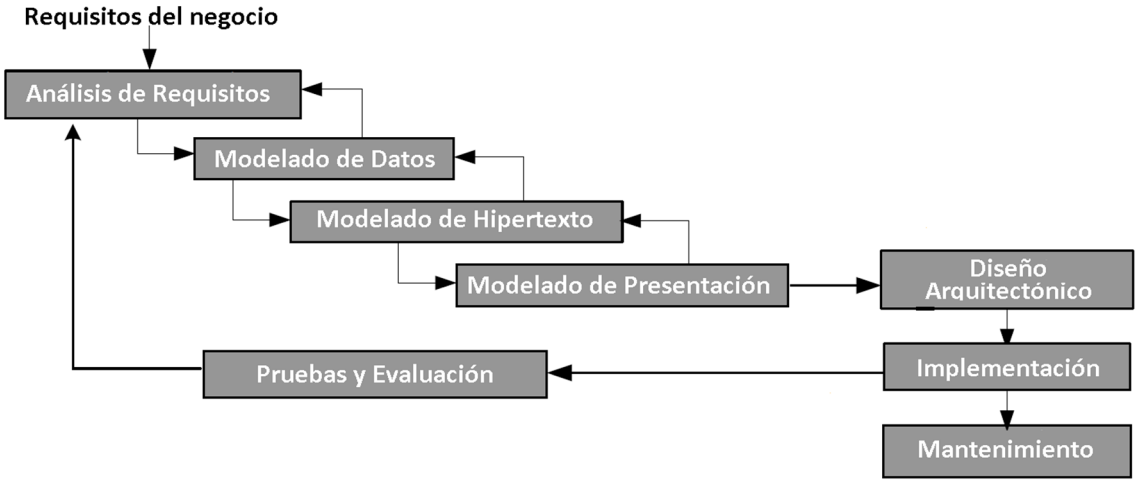
Los modelos WebML son modelos conceptuales de alto nivel, y, por tanto independientes de detalles de implementación que pueden ser resueltos por miembros no técnicos de un equipo de desarrollo. Sin embargo el enfoque cubre tanto la etapa de diseño como la de implementación, generando aplicaciones Web listas para ser ejecutadas. Para esto, WebML requiere de su propio entorno de desarrollo llamado WebRatio[1], el cual actúa

[1] https://www.webratio.com.

como una herramienta CASE propietaria que transforma los modelos abstractos en código ejecutable.

El proceso de desarrollo WebML consiste en diferentes etapas que se aplican de forma iterativa e incremental. En cada iteración se prueba y evalúa la aplicación, y de acuerdo a los resultados, puede ser extendida o modificada para cumplir con los nuevos o cambiantes requisitos. La Figura 2.1 muestra las etapas en el proceso de desarrollo WebML. Las etapas de modelado de datos, modelado de hipertexto y modelado de presentación comprenden el *modelado conceptual* que tiene especial interés desde el punto de vista de este trabajo.

Figura 2.1: Proceso de desarrollo de WebML



Estas tres etapas comprenden respectivamente el diseño de tres modelos ortogonales: modelo de datos, el cual es una extensión del modelo entidad-relación; modelo de hipertexto, el cual modela mapas navegacionales junto con la composición de las páginas; y el modelo de presentación, que describe el aspecto visual de las páginas incluyendo *layout* y *look & feel.* Aunque los tres modelos pueden ser desarrollados en WebRatio de forma simple e intuitiva a través de notaciones visuales, tienen una rigurosa semántica que les permite ser transformados en código ejecutable. Por la relevancia que tiene para este trabajo, en la siguiente sección se describe el modelo de hipertexto.

2.2.1 El modelo de hipertexto en WebML

El modelo de hipertexto WebML se construye a partir del modelo de datos. Este modelo describe cómo será publicada la información en la aplicación Web. Conceptualmente, puede verse como compuesto de dos modelos: el *modelo de composición* y el *modelo de navegación*.

El modelo de composición define qué páginas estarán contenidas en la aplicación Web, el contenido de cada página y los procesos de negocio que pueden ser ejecutados por la interacción de los usuarios. Los elementos del modelo de composición comprenden las Unidades de Contenido (*Content Units*) y las Unidades Operacionales (*Operation Units*).

Las Unidades Operacionales se utilizan para modelar la ejecución de lógica de negocio lanzada por la interacción de los usuarios. Las Unidades de Contenido modelan el contenido de cada página y están inspiradas en las primitivas de acceso RMM (Isakowitz et al., 1995): son primitivas que, incluidas en modelos navegacionales, dan acceso a entidades del modelo de datos. Las principales unidades de contenido se detallan a continuación:

- Unidad de Datos (*Data Unit*): Unidad que da acceso a una entidad del modelo de datos junto con sus propiedades. Es decir, la unidad está definida por una entidad del modelo relacional y los campos que la componen. La Unidad Multidatos (*Multidata Unit*) presenta varias instancias de una entidad simultáneamente, de acuerdo a algún criterio de selección.
- Unidad Índice (*Index Unit*): representa a un conjunto de referencias a entidades del modelo de datos, donde cada referencia presenta una descripción de la entidad a la que apunta. Pueden ser vistos como una lista de objetos (entidades o instancias de componentes), sin presentar información detallada de cada objeto.
- Unidad de Desplazamiento (*Scroller Unit*): representa la visualización secuencial de un conjunto de entidades del modelo de datos; es más conocida como visita guiada (*Guided Tour*). Muestra comandos para acceder a los elementos del conjunto ordenado de objetos (primero, último, anterior, siguiente, etc.).
- Unidad de Entrada (*Entry Unit*): representa un formulario de entrada de datos vinculados a una entidad del modelo de datos.
- Página (*Page*): es una abstracción de una región auto-contenida de la interfaz de usuario. La granularidad de las unidades anteriores puede ser demasiado fina para los requisitos de composición de una aplicación, los cuales normalmente demandan que la información contenida en algunas entidades sea entregada junta. Para cumplir con este requisito, WebML proporciona la noción de página.

El modelo de navegación especifica la forma en la cual unidades de contenido y unidades operacionales son enlazadas a través de enlaces conceptuales llamados *links*. Estos,

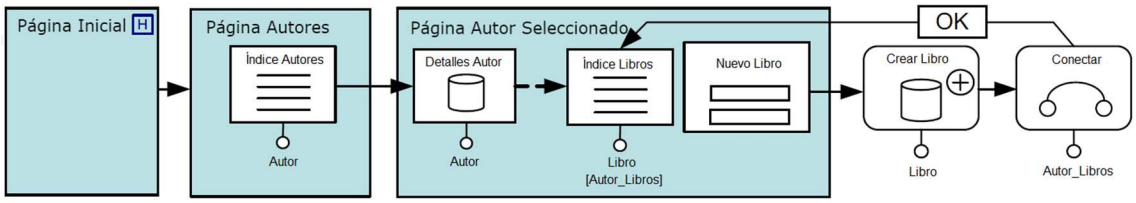
pueden ser de cinco tipos: normales (*normal links*), representados por flechas sólidas, indican navegación iniciada por usuarios; de transporte (*transport links*), representados por flechas discontinuas, indican movimientos de parámetros sin navegación; automáticos (*automatic links*), son enlaces normales pero automáticamente navegados por el sistema; y *OK links* y *KO links*, los cuales denotan respectivamente que la ejecución de la lógica de negocio tuvo éxito o fracaso. Estos enlaces pueden ser además de dos clases: contextuales, cuando relacionan a dos unidades llevando alguna información (llamada contexto) desde la unidad fuente a la unidad destino; y no-contextuales, cuando relacionan páginas de forma arbitraria, independientemente de las unidades que contengan y de la relación semántica entre estas unidades y el modelo de datos.

La Figura 2.2 muestra un ejemplo de un modelo de hipertexto en WebML que modela la navegación desde la entidad *Autor* a la entidad *Libro*, permitiendo además la creación de un *Libro*. Desde una *Página Inicial*, los usuarios pueden navegar a la *Página Autores* que muestra un índice de todos los autores registrados en una base de datos a través de la Unidad Índice llamada *Índice Autores*. La navegación se modela a través de un enlace normal no contextual debido a que no transporta información y simplemente permite un cambio de página.

La unidad *Índice Autores* permite a los usuarios seleccionar un autor y navegar a la *Página Autor Seleccionado*, la cual muestra los detalles de la entidad a través de una unidad de datos llamada *Detalles Autor*. Nótese que las unidades *Índice Autores* y *Detalles Autor* están relacionadas con la entidad *Autor* del modelo de datos, y que el enlace entre ellas es normal y contextual dado que transporta el identificador de la entidad seleccionada. La unidad *Detalles Autor* está asociada además a otra unidad índice llamada *Índice Libros*, la cual muestra todos los libros relacionados al autor actual, a través de un enlace de transporte modelando el transporte de información sin intervención del usuario y por tanto no generando navegación.

El ejemplo muestra también una unidad de entrada llamada *Nuevo Libro* que permite recolectar información para la creación de una entidad *Libro* vinculada al autor actual a través de las unidades operacionales *Crear Libro* y *Conectar*, las cuales se ejecutan secuencialmente creando la entidad libro y estableciendo la relación con el autor actual. El enlace entre las unidades *Conectar* e *índice Libros* es de tipo *OK Link* indicando que después de la ejecución exitosa de la lógica de negocio se retorna a la *Página Autor Seleccionado.*

Figura 2.2: Modelo de hipertexto en WebML.



Como puede verse en la Figura 2.2, cada tipo de componente conceptual se distingue a través de un icono, puede estar relacionado a una entidad del modelo de datos y puede especificar predicados (llamados *selectors*) para expresar restricciones o condiciones sobre las entidades relacionadas. Los componentes conceptuales pueden ser vistos como poderosos elementos de modelado que unidos a rigurosas reglas que aseguran el desarrollo de mapas navegacionales bien formados, permiten inferir el flujo de datos entre componentes y, en última instancia, el *workflow* ejecutable representado por el modelo de hipertexto.

WebRatio implementa el *algoritmo de computación de páginas* (Comai, S., & Fraternali, P. 2001) especificado por WebML que permite transformar los modelos de hipertexto en código ejecutable. El algoritmo describe como se infiere el contenido de las páginas después de cada evento navegacional representado por los diferentes tipos de enlaces. Debido a que una página WebML puede contener múltiples unidades enlazadas por medio de diferentes tipos de enlaces formando grafos de complejidad arbitraria, puede ser fácil generar modelos no deterministas o no computables debido a dependencias circulares entre unidades o páginas que WebRatio no sea capaz de resolver. En cualquier caso, se puede afirmar que el algoritmo de computación de páginas implementado en WebRatio actúa como un generador propietario de código (*black-box wizard*) que oculta la complejidad inherente a la generación de código desde modelos conceptuales.

2.2.2 El mecanismo de extensión de WebML

Como se comentó anteriormente, WebML fue diseñado para modelar aplicaciones Web que hacen uso intensivo de datos. Sin embargo, dado que las aplicaciones Web han evolucionado convirtiéndose en aplicaciones empresariales, WebML ha sido extendido para modelar algunos aspectos de este tipo de aplicaciones, aportándoles además los beneficios del modelado conceptual de alto nivel y la generación automática de código.

El mecanismo de extensión de WebML se basa en tres conceptos: (i) diseño basado en componentes, el cual permite extender el lenguaje sin alterar su semántica; (ii) uso de

diferentes tipos de enlaces, los cuales permiten conectar los componentes; y (iii) inferencia del contenido transportado por los enlaces, lo cual evita la especificación explicita de los datos transportados por ellos. Estos tres conceptos permiten extender WebML con componentes de cualquier tipo diseñados por desarrolladores, para cubrir en teoría, cualquier aspecto del desarrollo Web empresarial. Es necesario por su puesto, enriquecer la implementación de WebRatio para soportar los nuevos componentes incluyendo los motores necesarios para generar código a partir de ellos.

Actualmente, las principales extensiones de WebML incluyen soporte para Arquitecturas Orientadas a Servicios (SOA) (Manolescu et al., 2005), soporte para aplicaciones RIA (Rich Internet Application) (Bozzon et al., 2006), integración con Procesos de Negocio (Brambilla et al., 2006) y Web Semántica (Brambilla et al., 2007), entre otros. En los siguientes párrafos se comenta brevemente las extensiones para SOA y RIA.

La extensión WebML para SOA permite separar el diseño de la aplicación del diseño de los servicios. La extensión define nuevas unidades de modelado que cubren los procesos de negocio ejecutados por los servicios, la publicación de los servicios y su invocación. Entre las unidades de modelado para la publicación de servicios están *Service View*, *Port* y *Response Unit*, los cuales representan respectivamente el contenedor del servicio, las operaciones disponibles y las respuestas del servicio. Entre las unidades de modelado para la invocación están *Request* y *Response*, los cuales se usan desde las unidades de contenido para indicar la invocación del servicio.

De igual forma, la extensión define los nuevos tipos de enlaces para conectar las unidades de contenido con las nuevas unidades del servicio, y la implementación del generador de código necesario para producir el código ejecutable correspondiente a los nuevos elementos de modelado.

La extensión WebML para RIA permite modelar ciertas características de este tipo de aplicaciones como localización en cliente o en servidor de las unidades de contenido y patrones de comunicación asíncrono entre cliente y servidor. La extensión especifica los atributos localización (*Location*) y duración (*Duration*), que pueden ser aplicados a los elementos del modelo de datos (*Entities*) y a los elementos del modelo de hipertexto (unidades de contenido y operacionales) para indicar respectivamente, si están localizados en el cliente o servidor y si son temporales o persistentes. Por ejemplo, una unidad de datos puede especificar que las entidades o la sentencia de filtrado aplicadas residen en el lado del cliente y con una duración temporal. Además el concepto de página

se extiende agregando el concepto de página cliente (*Client Page*) en la cual el contenido y la lógica son administrados del lado del cliente y los eventos generados por la interacción del usuario pueden ser procesados localmente o enviados al servidor. La extensión incluye el generador de código implementado en WebRatio que genera el código que se ejecuta del lado del cliente implementado sobre la plataforma OpenLaszlo (OpenLaszlo, 2010).

2.2.3 WebML y E-WAE

La diferencia fundamental entre WebML y E-WAE radica en que WebML es un lenguaje específico del dominio Web (*Domain Specific Language*, DSL) que provee elementos de modelado de alto nivel de abstracción para caracterizar los conceptos de ese dominio, mientras E-WAE es una notación que caracteriza la capa de presentación Web siguiendo la filosofía UML, y por tanto, proporcionando una semántica orientada a código para cada uno de sus elementos por medio de la definición de modelos PSM (obtenidos a partir de los modelos PIM).

Con WebML, los diseñadores no requieren grandes conocimientos técnicos de desarrollo Web porque el lenguaje les permite expresar de forma natural los conceptos de ese dominio, y la generación de código está gobernada por los elementos de modelado y su herramienta CASE. En este sentido, se puede afirmar que los elementos de modelado de alto nivel ocultan la complejidad inherente al desarrollo Web a expensas de acoplar la generación y mantenimiento de las aplicaciones a la notación misma y su herramienta generadora de código que actúa como un generador propietario de código (*black-box wizard*).

Dado que E-WAE es una notación UML y las transformaciones PIM a PSM y PSM a código están definidas de forma explícita, no se requiere de herramientas específicas para la generación de código, garantizando la portabilidad de los modelos. Además, dado que los modelos PSM definen una semántica orientada a código de los elementos definidos en E-WAE, facilita el seguimiento dentro del proceso MDD precisando los elementos en los cuales el código está basado y en última instancia cerrando la brecha entre modelos y código de forma transparente.

Aunque los elementos de modelado de WebML facilitan el desarrollo de los diseños, estos, y las aplicaciones generadas, están limitados por su poder expresivo. Así, WebML define elementos que tienen la capacidad de modelar la invocación de un servicio Web, y aunque estos pueden implícitamente implementar un conjunto predefinido de patrones

de diseño, no permiten la flexibilidad de adaptar o combinar los patrones para solucionar un problema de negocio específico. Por ejemplo WebML no puede modelar el patrón de seguridad *Secure Message Router* (Alur et al., 2003) para aplicar mecanismos de seguridad a nivel de mensajes, el patrón *Service Activator* (Alur et al., 2003) para describir la invocación asíncrona de los servicios o una combinación de los dos. Estos, como la mayoría de los patrones arquitectónicos y de diseño, se definen usando UML y por tanto pueden ser directamente integrados en los modelos propuestos por el enfoque E-WAE.

Otra diferencia importante entre WebML y E-WAE se relaciona con la arquitectura de las aplicaciones. E-WAE ha sido desarrollada con el concepto de la arquitectura multicapa en mente, centrándose sólo en la capa de presentación Web, la cual puede ser diseñada y generada de forma desacoplada del modelo de datos subyacente por medio de objetos de transferencia de datos (DTOs) (Fowler, 2002). A diferencia, WebML tiene la capacidad de diseñar y generar aplicaciones Web completas sin tener en cuenta la arquitectura multicapa, por ejemplo, el modelo de hipertexto no puede existir aisladamente del modelo de datos, los elementos de modelado que lo componen son en esencia primitivas de acceso al modelo de datos. En este sentido, E-WAE puede encajar en el desarrollo de aplicaciones empresariales basadas en la arquitectura multicapa, mientras WebML puede encajar en el desarrollo de aplicaciones dirigidas por datos, dado que sus elementos de modelado tienen un comportamiento predefinido basado en operaciones CRUD (*create, read, update, delete*).

Por último, a diferencia de E-WAE, WebML es una notación propietaria que no alcanza el nivel de aceptación y difusión de UML, por tanto, no puede interoperar con otras notaciones, ni integrar o ser integrada con otros modelos, dificultando la reusabilidad y convirtiéndose en un ecosistema de desarrollo aislado. Para intentar salvar estos obstáculos, WebML define un perfil UML que representa sus elementos de modelado (Moreno et al., 2007) con el objetivo de integrar su proceso de desarrollo en los entornos de desarrollo estándar.

Sin embargo, su perfil UML provee el mismo nivel de expresividad que WebML, cada concepto de alto nivel definido en WebML es traducido directamente a un estereotipo. Por ejemplo, cada unidad de contenido y cada unidad operacional se corresponde con un estereotipo que extiende la definición UML de componente, manteniendo así la filosofía de WebML: cada estereotipo representa un concepto de alto nivel del dominio Web. De hecho, la mayoría de estereotipos definidos en el perfil se aplican a componentes UML,

y los enlaces WebML se representan a través de elementos de conexión UML que relacionan estos componentes.

El hecho de que el perfil defina un alto número de estereotipos (más de 50) y que la semántica de los estereotipos no este dada en términos de código, hace que los diseñadores puedan utilizar herramientas UML CASE genéricas para desarrollar los modelos, pero no generar código a partir de ellos. El enfoque sugiere exportar los modelos extendidos con el perfil UML a WebRatio para generar la implementación de las aplicaciones.

Por tanto, a diferencia de E-WAE, que basa sus modelos en diagramas UML de clase y UML de secuencia, con clases directamente traducibles a código orientado a objetos, el perfil UML para WebML requiere de herramientas específicas para generar el código de las aplicaciones y sus elementos no pueden ser directamente integrados en modelos que implementen patrones arquitectónicos y de diseño, según las necesidades de los diseñadores.

2.3 UML-based Web Engineering - UWE

UML-based Web Engineering – UWE (Hennicker & Koch, 2001) comprende un proceso de desarrollo Web que promueve el diseño sistemático y la generación semiautomática de aplicaciones Web usando exclusivamente las técnicas, la notación y los mecanismos de extensión UML. UWE cubre completamente el proceso de desarrollo desde la especificación de requisitos hasta la generación de código. Sus autores hacen énfasis en que UML es lo suficientemente eficaz para cubrir todo el proceso de modelado Web. Para modelar los aspectos generales de las aplicaciones Web, UWE utiliza la notación y técnicas de UML estándar, mientras que para modelar los aspectos especiales de las aplicaciones Web, como navegación y presentación, UWE proporciona un perfil UML. De esta forma UWE puede aprovechar todo el conocimiento, experiencia y herramientas asociadas con UML. No obstante, UWE implementa su propia herramienta CASE.

2.3.1 El proceso de desarrollo UWE

El proceso de desarrollo UWE incluye análisis, diseño y generación de aplicaciones Web. Para los aspectos generales de las aplicaciones Web, utiliza el enfoque orientado a objetos, iterativo e incremental basado en UML y el Proceso Unificado de Desarrollo (RUP) (Jacobson et al., 1999). Las actividades básicas de modelado son el análisis de requisitos, diseño conceptual, diseño navegacional y diseño de presentación.

Para describir los requisitos funcionales, UWE usa los diagramas de casos de uso UML y diagramas de actividad UML. Koch & Kraus (2002) muestran como los requisitos funcionales de una aplicación Web pueden especificarse a través de estos dos tipos de diagramas. Los casos de uso se utilizan para describir como la aplicación interactúa con actores externos sin detallar la estructura interna. Los diagramas de actividad detallan el comportamiento interno de la aplicación especificando la secuencia de acciones que se ejecutan después de la interacción con los usuarios externos. Esto representa un desajuste importante con el Proceso Unificado de Desarrollo, que utiliza diagramas de actividades para especificar casos de uso, y diagramas de secuencia para detallar la implementación de estos casos de uso. Es decir, UWE utiliza diagramas de actividad para especificar la información que el Proceso Unificado de Desarrollo proporciona a través de diagramas de secuencia.

El modelo conceptual está basado en los requisitos funcionales y describe el conjunto de elementos estáticos del dominio. UWE representa este modelo a través de diagramas de clase UML intentando ignorar tanto como sean posible los aspectos navegacionales y de presentación de las aplicaciones Web. Estos aspectos son pospuestos hasta el diseño de sus respectivos modelos. Esta característica favorece la independencia del modelo del dominio de la plataforma Web favoreciendo la reusabilidad. Por la relevancia que tiene para este trabajo, en las siguientes secciones describimos el modelo navegacional y de presentación.

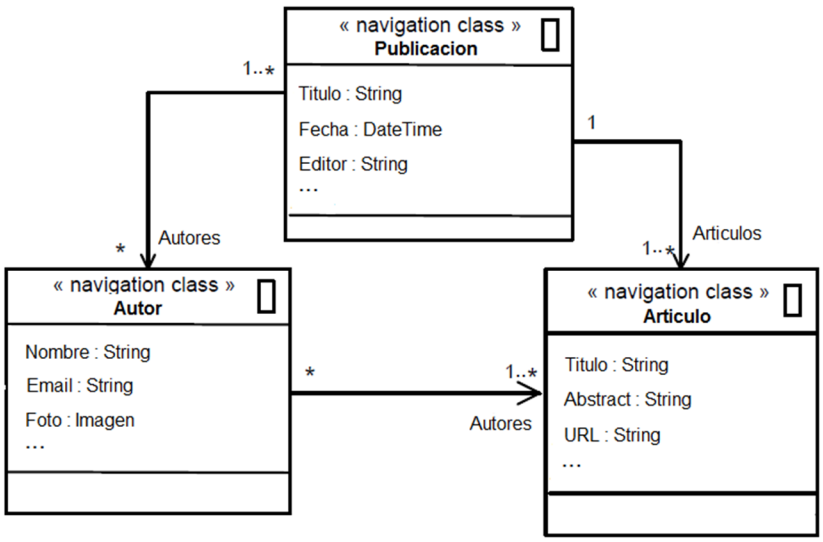
2.3.2 El modelo navegacional UWE

El modelo navegacional UWE comprende la construcción de dos modelos: *el modelo del espacio navegacional* y *el modelo de la estructura navegacional*. El primero especifica que entidades se pueden visitar por medio de la navegación Web, y el segundo especifica cómo se llega a estas entidades. Los dos modelos navegacionales se representan por medio de diagramas de clases extendidos con un perfil UML.

UWE proporciona un conjunto de directrices y mecanismos semiautomáticos para desarrollar el modelo navegacional partiendo del modelo conceptual (Hennicker & Koch, 2000). El primer paso consiste en seleccionar las entidades del modelo conceptual que deben ser alcanzables por medio de la navegación Web y a través de que caminos navegacionales. Esta decisión debe estar basada en los requisitos que la aplicación debe satisfacer.

Para la construcción del modelo del espacio navegacional, el perfil UWE define dos estereotipos principales: clase navegable (*navigation class*) y navegación dirigida (*direct navigability*). El primero se aplica a las clases del modelo conceptual que pueden ser alcanzadas mediante la navegación Web. El segundo se aplica a las asociaciones dirigidas que relacionan estas clases modelando la navegabilidad dirigida desde la clase fuente a la clase destino, no obstante estas asociaciones pueden ser bidireccionales. La Figura 2.3 muestra un ejemplo de un modelo del espacio navegacional de una aplicación. Cabe destacar que sólo las clases del modelo conceptual que son relevantes para modelar los aspectos navegacionales se incluyen en este modelo.

Figura 2.3: Modelo del espacio navegacional en UWE.



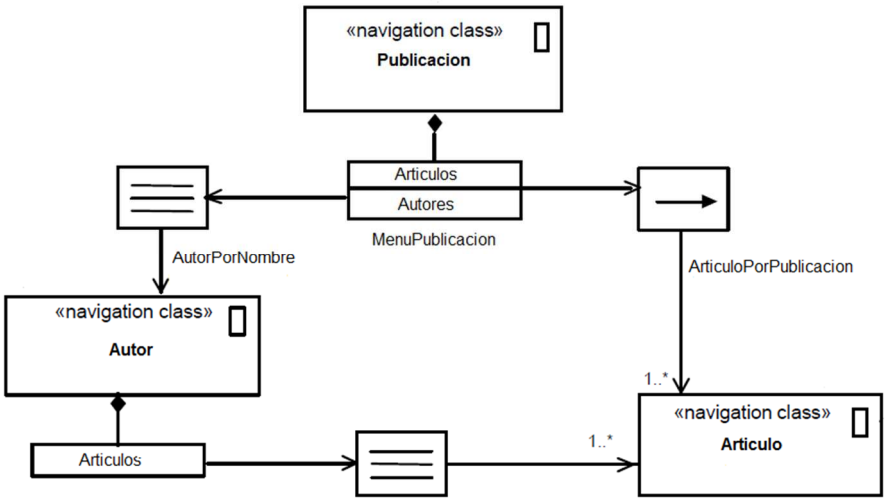
El modelo de la estructura navegacional se construye a partir del modelo del espacio navegacional, y puede considerarse como un paso de refinamiento en el proceso de desarrollo con UWE, en el cual se agregan elementos de acceso encargados de desencadenar la navegación Web. Los principales elementos de acceso, llamados primitivas de acceso, se representan con los siguientes estereotipos:

- Índice (*Index*): comprende la descripción de una lista de instancias de una clase navegada, permitiendo el acceso directo a cada elemento de la lista desde la interfaz de usuario.
- Visita guiada (*Guided Tour*): elemento que da acceso a la primera instancia de una clase navegada permitiendo el acceso secuencial al resto de instancias. Las visitas guiadas pueden ser controladas por los usuarios o por el sistema.

- Menú (*Menu*): puede ser visto como un índice de alto nivel que da acceso a conjuntos de clases navegadas. Cada aplicación Web tiene al menos un punto de entrada o nodo inicial llamado menú principal.

La Figura 2.4 muestra el modelo de la estructura navegacional basado en el modelo del espacio navegacional de la Figura 2.3, como puede observarse, este tipo de diagrama resulta muy útil en la representación estática de la estructura navegacional. Este diagrama muestra como navegar a través de las entidades navegables por medio del uso de los elementos de acceso anteriormente enumerados. El siguiente paso en el proceso de desarrollo UWE es describir como se presenta al usuario la información del modelo de la estructura navegacional. Esto se describe a través del modelo de presentación UWE, el cual muestra el diseño de las interfaces de usuario de forma abstracta, centrándose en la organización estructural de la presentación y no en la apariencia física.

Figura 2.4: Modelo de la estructura navegacional en UWE.



2.3.3 El modelo de presentación UWE

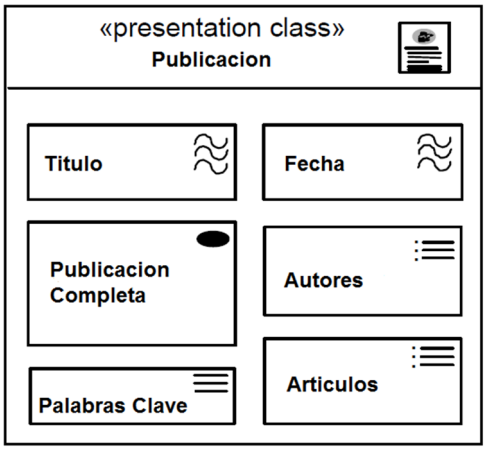
El modelo de presentación UWE describe dónde y cómo las entidades que pueden ser alcanzadas por medio de la navegación serán presentadas al usuario. En UWE, los diseños de presentación se construyen por medio de transformaciones semiautomáticas del modelo de la estructura navegacional en un conjunto de modelos que muestran la localización estática de las entidades visibles al usuario.

El modelo de presentación UWE se describe a través de un tipo especial de diagrama UML de clases que permite representar las interfaces de usuario describiendo el orden espacial y dimensiones relativas de sus componentes. Aunque estas características no

pueden ser administradas por las herramientas UML CASE genéricas, pueden ser útiles porque constituyen esquemas de las interfaces de usuarios que facilitan la comunicación con los usuarios finales. Por tanto, el modelo de presentación proporciona una vista de la interfaz de usuario que describe la estructura básica pero que abstrae de aspectos concretos como *look & feel*.

Para la construcción del modelo de presentación, el perfil UWE define dos estereotipos principales: clase de presentación (*presentation class*) y contenedor (*frameset*). El primero se aplica a clases UML que se corresponden directamente con las clases del modelo navegacional y pueden estar compuestas por elementos de presentación básicos, los cuales son representados por los estereotipos *text*, *button*, *image*, *form*, etc. El segundo representa contenedores que pueden estar compuestos por otros contenedores anidados y por clases de presentación. La Figura 2.5 muestra un ejemplo de una clase de presentación para la entidad *publicación* referida en las Figuras 2.3 y 2.4.

Figura 2.5: Modelo de presentación en UWE.



2.3.4 El entorno de desarrollo UWE

Como se comentó anteriormente, UWE utiliza la notación y técnicas UML estándar. Por tanto, cualquier herramienta UML CASE genérica puede ser usada para desarrollar los modelos UWE. Sin embargo, ciertas características particulares que dan valor agregado a los modelos, como por ejemplo, la capacidad de describir la disposición de los componentes en los modelos de presentación y la generación automática de código, no pueden ser soportadas por herramientas UML CASE genéricas.

Para soportar estas características particulares, UWE requiere de editores y generadores de código específicos para su notación. El proceso de desarrollo UWE promueve el

desarrollo dirigido por modelos (MDD) en todas las fases del desarrollo, desde la especificación de requisitos a los modelos de diseño y desde estos a la implementación. Sin embargo, dado que los modelos navegacionales y de presentación utilizan elementos de alto nivel de abstracción que caracterizan los conceptos del dominio Web sin una semántica orientada a código, se requiere de herramientas específicas para generar código a partir estos elementos abstractos (*black-box wizards*). Para satisfacer estos requisitos, UWE proporciona su propio *plug-in* llamado ArgoUWE que se integra en la herramienta CASE ArgoUML. ArgoUWE permite aprovechar todas las características de UWE incluida la generación (semi)automática de código y la verificación de consistencia de los modelos de acuerdo a las restricciones OCL especificadas en su meta-modelo.

2.3.5 Herramienta CASE UWE4JSF

Kroiss et al. (2009) presentan UWE4JSF, otra herramienta CASE desarrollada para implementar el desarrollo dirigido por modelos (MDD) de aplicaciones Web diseñadas con UWE. UWE4JSF se integra completamente en el IDE Eclipse, reutilizando sus editores de modelos y tecnologías de transformación para generar automáticamente aplicaciones Web para la plataforma JSF. Las transformaciones entre modelos se implementan utilizando el lenguaje ATL[2], y la generación de código se implementa utilizando el generador de código JET[3].

Para implementar la generación de código automática, UWE4JSF requiere que los modelos UWE sean extendidos con información específica, como la descripción de los datos de entrada de las clases navegadas en el modelo de la estructura navegacional, o la descripción concreta de los elementos de la interfaz de usuario en el modelo de presentación que los convierte en modelos de presentación concreta (*concrete presentation model*).

El proceso toma como entrada los modelos UWE junto con la información que los extiende, los cuales son validados usando un conjunto de restricciones OCL y transformados a modelos específicos de la plataforma JSF (PSM). Los modelos PSM son finalmente utilizados como entrada en el proceso de generación de código, el cual genera clases Java, especificaciones de las páginas y ficheros de configuración. El código

[2] http://www.eclipse.org/atl/

[3] http://www.eclipse.org/modeling/m2t

generado debe ser complementado con código que implemente la lógica de negocio de la aplicación.

2.3.6 UWE y E-WAE

La principal similitud entre UWE y E-WAE radica en que son enfoques basados en UML. Esta característica hace que los dos enfoques puedan aprovechar las ventajas del estándar UML como la capacidad de integrarse con otros modelos y ser generados y mantenidos con herramientas UML CASE genéricas.

UWE cubre el proceso de desarrollo de aplicaciones Web completas desde la especificación de requisitos hasta la generación de código. Para tratar con el modelado de los aspectos generales de software, UWE sigue la filosofía UML: proporciona una semántica orientada a código para cada uno de sus elementos con clases UML directamente traducibles a código orientado a objetos. Sin embargo, para tratar con los aspectos específicos de las aplicaciones Web, como el modelado de los aspectos navegacionales y de presentación, UWE sigue la filosofía de los lenguajes específicos de dominio como WebML: define elementos de modelado de alto nivel de abstracción que caracterizan conceptos del dominio Web.

Aunque estos elementos se definen a través de un perfil UML que les permiten ser mantenidos con herramientas UML CASE genéricas, los estereotipos tienen una semántica de alto nivel muy concreta (Berner et al., 1999) que no les permite ser traducidos directamente a código. Dado que UWE no proporciona una semántica orientada a código para estos elementos, se requiere de herramientas específicas como ArgoUWE para generar la implementación de los aspectos particulares de las aplicaciones Web que representan.

E-WAE se centra en el diseño e implementación de la capa de presentación Web manteniendo la filosofía de UML y por tanto proporcionando una semántica orientada a código para cada uno de sus elementos por medio de la definición de modelos PSM. De esta forma, se facilita la trazabilidad en el proceso MDD, precisando los elementos en los cuales el código está basado, y, en última instancia, cerrando la brecha entre modelos y código de forma transparente. Para el diseño y desarrollo de las capas de negocio y de integración E-WAE, al igual que UWE, promueve la utilización de UML estándar, aunque UWE utiliza diagramas de actividades para representar la información que el Proceso Unificado de Desarrollo caracteriza con diagramas de secuencia.

Los elementos de acceso utilizados en los diagramas de navegación UWE son primitivas de acceso RMM (Isakowitz et al., 1995) que dan acceso a entidades del modelo conceptual. Esta dependencia del modelo navegacional en el modelo conceptual dificulta el desarrollo de aplicaciones de acuerdo a la arquitectura multicapa que promueve la independencia de la capa de presentación del resto de capas de una aplicación. Esta característica unida la semántica de alto nivel de los elementos de acceso UWE impide la incorporación en los modelos de patrones multicapa que facilitan la mantenibilidad de una aplicación. E-WAE, en línea con la arquiterctura multicapa, promueve el desarrollo de la capa de presentación de forma desacoplada del resto de capas, a través del uso de *Objetos de Transferencia de Datos* (DTO), el mecanismo básico para intercambiar información entre capas en esta arquitectura. Esta característica permite generar una capa de presentación Web completamente funcional sin requerir la definición de objetos de negocio ni ejecución de lógica de negocio.

El modelo de presentación UWE puede generar esquemas de la interfaz de usuario que muestran la disposición de sus elementos, lo que facilita la comunicación con los usuarios finales, al coste de requerir herramientas CASE propietarias. El enfoque E-WAE, siguiendo la filosofía MDD, tiene la capacidad de generar mockups interactivos con los que los usuarios pueden interactuar, para validar o refinar los requisitos en el campo. Sin embargo, estos mockups interactivos se centran en el aspecto navegacional de las aplicaciones Web, y no tienen en cuenta la disposición de los elementos de la interfaz de usuario ni el *look & feel*.

Aunque la extensión UWE4JSF define modelos PSM para la plataforma JSF, estos y las reglas de transformación que implementa, están basados en la plataforma Eclipse: EMF (Eclipse Modeling Framework) para implementar los modelos PSM, ATL para implementar las transformaciones PIM a PSM, y JET para implementar las transformaciones PSM a código. De esta forma, aunque la contribución más importante de esta extensión es la generación de código automáticamente desde modelos PSM, la falta de una semántica orientada al código hace que el proceso dependa de herramientas propietarias a las que se vincula el mantenimiento del código.

Dado que los perfiles definidos por E-WAE están basados en UML y las reglas de transformación PIM a PSM y PSM a código están definidas de forma explicita, no dependen de herramientas específicas. Además dada la semántica orientada a código de los elementos que define el enfoque E-WAE, no es difícil generar otros perfiles UML

para modelos PSM con sus respectivas transformaciones que soporten otras plataformas de desarrollo Web.

2.4 Object-Oriented Hypermedia – OOH

Object-Oriented Hypermedia – OOH (Gómez et al., 2001) propone un método que proporciona a los diseñadores una notación y semántica específicas para modelar interfaces Web. OOH está basado en el método OOM (Pastor et al., 1997), el cual define un proceso de desarrollo que hace énfasis en el modelado conceptual conforme con la metodología UML siguiendo el paradigma de desarrollo orientado a objetos. Esta base UML hace suponer que OOH, al igual que UWE, es adecuado para el diseño y desarrollo de aplicaciones Web dinámicas, con alta complejidad, donde la modularidad y la consistencia son factores clave (Cachero, 2003).

OOH permite a los diseñadores centrarse en los conceptos necesarios para modelar interfaces Web compatibles con los modelos UML generados previamente con el método OOM. Para esto, OOH define dos nuevos tipos de diagramas principales a nivel conceptual para cubrir los aspectos navegacionales y de presentación de las aplicaciones Web, llamados respectivamente, *Diagrama de Acceso Navegacional* (NAD) y *Diagrama de Presentación Abstracta* (APD). Estos dos tipos de diagramas, con el soporte de un conjunto de patrones definidos en el *Catálogo de Patrones de Interfaz*, propuestos también por el método OOH, tienen la capacidad de modelar una interfaz Web completa. Además, OOH implementa una herramienta CASE propietaria que le permite generar aplicaciones listas para ser ejecutada desde estos diagramas. Aunque estos dos tipos de diagrama se basan en los diagramas desarrollados con el método OOM, no son diagramas UML pues la notación no sigue sus mismas reglas sintácticas y semánticas.

2.4.1 El proceso de desarrollo OOH

El proceso de desarrollo OOH comienza definiendo los diagramas de acceso navegacional, los cuales extienden con características navegacionales a los diagramas UML de clases que representan el modelo de dominio, desarrollados con el método OOM. OOH tiene en cuenta los diferentes roles de usuarios presentes en el dominio de la aplicación. Dado que cada rol tiene una vista diferente del sistema, es necesario definir para cada uno un diagrama de acceso navegacional diferente.

Una vez que se construyen los diagramas de acceso navegacional, se puede generar a partir de ellos, siguiendo la filosofía MDD, una interfaz Web por defecto. Esta interfaz

por defecto, por su bajo nivel de sofisticación, tanto desde el punto de vista visual como de usabilidad, suele considerarse un prototipo. Para mejorar la calidad de esta interfaz, OOH introduce el segundo tipo de diagrama, el diagrama de presentación abstracta, cuya estructura por defecto se deriva de los diagramas de acceso navegacional. Para refinar la estructura de presentación, OOH propone el catálogo de patrones de interfaz que definen un conjunto de soluciones que han probado ser efectivas para ciertos problemas identificados en la interacción entre usuarios y aplicaciones Web. Cada patrón del catálogo describe un problema particular y varias implementaciones de la solución, permitiendo al diseñador seleccionar la más adecuada para la aplicación.

Una vez se refinan los diagramas de presentación abstracta, la interfaz de la aplicación Web se puede generar de forma automática por medio de la herramienta CASE propietaria para diferentes entornos (HTML, WML, ASP, JSP, etc.) produciendo aplicaciones listas para ser ejecutadas. La herramienta CASE proporciona un entorno de desarrollo que soporta toda la metodología de OOH, simplificando el diseño y la implementación de aplicaciones Web por medio de editores y generadores de código para los diagramas propuestos por OOH. Por la relevancia que tienen para este trabajo, en las siguientes secciones se describe el diagrama de acceso navegacional y el diagrama de presentación abstracta.

2.4.2 Diagrama de acceso navegacional OOH

El modelo navegacional en OOH se representa a través de uno o más diagramas de acceso navegacional. OOH sugiere construir tantos diagramas navegacionales como vistas del sistema sean requeridas. Por regla general, se requiere al menos un diagrama de acceso navegacional por cada rol de usuario en el dominio de la aplicación.

El desarrollo de un diagrama de acceso navegacional implica filtrar y enriquecer los diagramas UML de clases que modelan el dominio de la aplicación desarrollados a través de método OOM durante la fase de modelado conceptual. Este proceso requiere tener en cuenta aspectos como las entidades que pueden ser visitadas a través de la navegación, el orden, la cardinalidad y la relación de estas con los roles de usuarios. Un diagrama de acceso navegacional se basa en los siguientes cuatro tipos de constructores:

- Clases navegacionales (*Navigational Classes*): entidades basadas en las clases UML del modelo conceptual. Pueden considerarse como clases del dominio cuya visibilidad de sus propiedades y métodos han sido enriquecidos de acuerdo a los requisitos navegacionales. Un ejemplo de enriquecimiento es la aplicación de tres

tipos de atributos: atributos V (atributos siempre visibles), atributos R (atributos referenciados a los cuales se tendrá acceso bajo demanda del usuario) y atributos H (atributos ocultos).

- Destinos navegacionales (*Navigational Targets*): conjuntos de clases navegacionales que proporcionan juntas la implementación de un requisito funcional.
- Enlaces navegacionales (*Navigational Links*): definen los caminos navegacionales conectando clases navegacionales. OOH define cuatro tipos de enlaces: enlaces I (enlaces internos que definen el camino navegacional dentro de los límites de un destino navegacional); enlaces T (enlaces transversales entre clases navegacionales que perteneces a diferentes destinos navegacionales); enlaces R (enlaces de requisito que indican el punto de entrada de la navegación en un destino navegacional); y enlaces S (enlaces de servicio que definen la conexión con ejecución de lógica de negocio).
- Colecciones (*Collections*): se representan por medio de un triángulo invertido. Modelan cómo se agrupa la información de la interfaz de usuario para facilitar el acceso del usuario final.

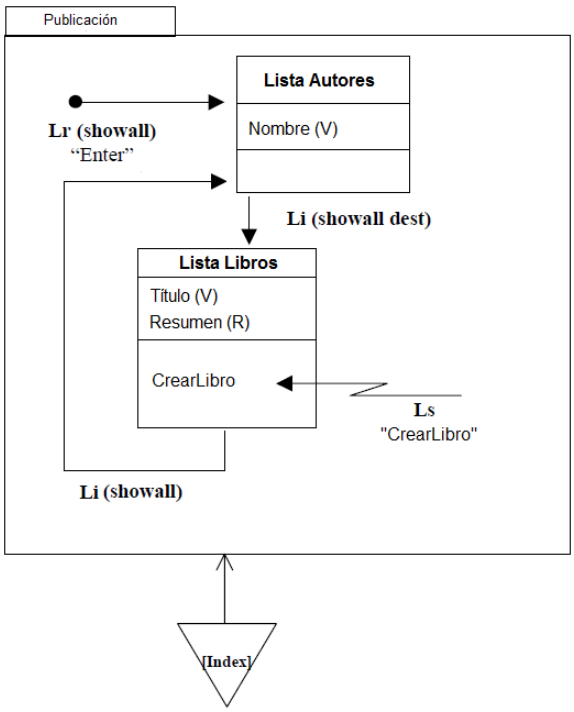
La Figura 2.6 muestra un ejemplo de un diagrama de acceso navegacional que modela un requisito navegacional que permite a los usuarios navegar a una lista de entidades *Autor*, seleccionar un autor y navegar a las entidades *Libro* relacionadas al autor seleccionado, permitiendo además crear nuevas entidades libro y retornar a la lista de autores. El destino navegacional llamado *Publicación* agrupa las entidades, servicios y relaciones necesarios para implementar el requisito navegacional.

El destino navegacional *Publicación* está compuesto por las vistas de las clases de dominio *Autor* y *Libro*. La primera tiene una sola propiedad llamada *Nombre* a la que se le ha aplicado el atributo V (siempre visible) indicando que debe estar siempre presente en la vista. La clase *Libro* tiene la propiedad *Resumen* con el atributo R (referenciado), lo cual implica que el usuario tiene que explícitamente navegar para acceder al resumen del libro.

También se puede observar en la Figura 2.6 que las clases navegacionales *Autor* y *Libro* se relacionan a través de un enlace I (enlace interno) que tiene asociado el patrón navegacional *showall*, lo cual causa que todas las instancias de la clase *Libro* aparezcan juntas. El modificador *dest*, aplicado al patrón navegacional, indica que el enlace implica

un paso en el proceso navegacional, es decir, que la lista de autores y la lista de libros se mostraran en diferentes páginas. El enlace R (enlace de requisito) etiquetado con *Enter* apunta a la clase navegacional que iniciará el proceso que cumple con el requisito del destino navegacional. Finalmente, el enlace S (enlace de servicio) se asocia con el servicio *Crear*, lo cual modela la funcionalidad que permite a los usuarios crear una nueva entidad *Libro*.

Figura 2.6: Diagrama de Acceso Navegacional en OOH.



2.4.3 Diagrama de presentación abstracta OOH

Como se comentó, los diagramas de presentación abstracta pueden derivarse de la información capturada en los diagramas de acceso navegacional en términos de plantillas *por defecto* que especifican la apariencia visual, así como la estructura de las páginas Web. OOH define cinco tipos de plantillas:

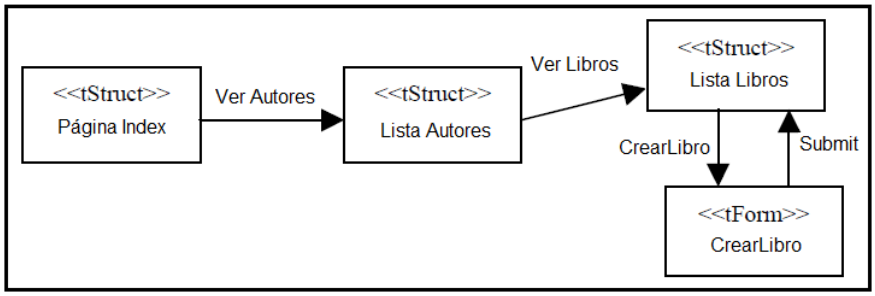
- Plantilla de estructura (*tStruct*): plantilla para definir la información que aparecerá en la página abstracta.
- Plantilla de estilo (*tStyle*): plantilla para definir las características tales como ubicación física de los elementos, tipografía o paleta de colores.
- Plantilla de formulario (*tForm*): plantilla para definir qué elementos de datos son requeridos desde el usuario para interactuar con el sistema.

- Plantilla de funcionalidad (*tFunction*): plantilla para definir la funcionalidad del lado del cliente.
- Plantilla de ventana (*tWindow*): plantilla para definir conjuntos de vistas simultaneas disponibles para un usuario.

Gomez et al. (2001) muestran los pasos definidos en OOH para traducir los diferentes elementos contenidos en los diagramas de acceso navegacional a las diferentes plantillas que constituyen los diagramas de presentación abstracta, de los cuales se puede obtener una simple pero funcional interfaz Web. Normalmente esta interfaz Web requiere refinamiento para ser incluida en una aplicación final, y su verdadero valor está en servir como prototipo en el cual validar que los requisitos de usuario estén correctamente capturados.

El proceso de refinamiento de los diagrama de presentación abstracta consiste en la modificación de su estructura. La forma más fácil de llevarlo a cabo es mediante la aplicación de los patrones definidos en el catálogo de patrones de OOH. En la Figura 2.7 se muestra el diagrama de presentación abstracta que se corresponde con la plantilla de estructura por defecto derivada desde el diagrama de acceso navegacional presentado en la Figura 2.6.

Figura 2.7: Diagrama de Presentación Abstracta en OOH.



2.4.4 OOH4RIA

Melia et al. (2008) presentan OOH4RIA como un proceso de desarrollo de software dirigido por modelos que pretende cubrir las etapas de diseño y desarrollo de aplicaciones RIA (*Rich Internet Application*). OOH4RIA se basa en una extensión y mejora de la metodología OOH, a la que se ha agregado los modelos y transformaciones necesarias para obtener aplicaciones RIA.

OOH4RIA reutiliza el modelo conceptual y el modelo navegacional de OOH para generar, por un lado, la implementación del lado del servidor y, por el otro, dos nuevos modelos específicos para la plataforma GWT (*Google Web Toolkit*)[4] llamados *modelo de presentación* y *modelo de orquestación*, los cuales son utilizados para generar la implementación del lado del cliente. De esta forma OOH4RIA implementa transformaciones PIM a código, para generar el código del lado del servidor, PIM a PSM para generar los modelos que representan el lado del cliente, y PSM a código para generar el código del lado del cliente. Para facilitar y acelerar el desarrollo de los modelos y la aplicación de estas transformaciones, OOH4RIA implementa su propia herramienta CASE llamada *OOH4RIA Tool* (Melia et al., 2009) desarrollada como un *plug-in* del entorno Eclipse.

El primer paso del proceso de desarrollo OOH4RIA comprende la transformación del modelo de navegación OOH en el modelo de presentación PSM, el cual describe la estructura estática de la interfaz Web por medio de diferentes *widgets* implementados en la plataforma GWT. Este modelo requiere de la intervención manual para definir la localización espacial y el estilo de los *widgets*. Dado que los *widgets* en una interfaz RIA requieren de contenido dinámico, es necesario especificar la interacción de estos elementos con los componentes del lado del servidor que les permiten descargar este contenido de forma asíncrona. Esta interacción se describe a través del modelo de orquestación, que se representa a través de un diagrama de estados UML extendido con un perfil UML. Este modelo se obtiene a partir del modelo de navegación OOH y del modelo de presentación PSM y también tiene que ser completado manualmente introduciendo los eventos y las operaciones en los diferentes estados.

Para soportar este proceso de desarrollo, la herramienta OOH4RIA Tool, se fundamenta en el framework GMF para definir la parte de modelado, y en el framework OpenArchitectureWare para la implementación de las transformaciones. Para soportar el desarrollo de los modelos OOH4RIA, se ha definido un meta-modelo Ecore que recoge todos los conceptos de los modelos y las relaciones entre ellos. Además se ha implementado un conjunto de restricciones OCL en los editores gráficos para asegurar que los modelos generados estén bien formados. Respecto a las transformaciones,

[4] www.gwtproject.org

OOH4RIA Tool permite ejecutarlas y modificarlas para que los desarrolladores puedan adaptarlas para solucionar problemas específicos de su dominio.

2.4.5 OOH y E-WAE

Tanto OOH como E-WAE han sido desarrolladas con la idea de soportar el desarrollo de aplicaciones Web empresariales. Para intentar cumplir con este objetivo, OOH extiende un método basado completamente en UML que reutiliza todos sus beneficios, incluyendo el conocimiento, experiencia y herramientas asociadas a éste. Sin embargo la extensión propuesta por OOH incluye dos tipos de modelos que, aunque se basan en el modelo UML del dominio, no son UML. Estos dos tipos de modelos convierten a OOH en una notación propietaria que requiere de su propia herramienta CASE para desarrollarlos. Además, dado que los modelos propuestos son modelos conceptuales de alto nivel de abstracción sin una semántica orientada a código, la herramienta propuesta por OOH actúa como un generador de código propietario (*black box wizard*) que transforma los modelos abstractos en código ejecutable y vincula el manteniemento del código a dicho generador.

Por otro lado, E-WAE permite modelar la capa de presentación Web manteniendo la notación y filosofía de UML y por tanto proporcionando una semántica orientada a código para cada uno de sus elementos por medio de la definición de modelos PSM. Además, los modelos PSM facilitan el seguimiento dentro del proceso MDD precisando los elementos en los cuales se basa el código, y, en última instancia, cerrando la brecha entre modelos y código de forma transparente.

Aunque los modelos conceptuales OOH son fáciles de desarrollar, porque expresan de forma natural los conceptos del dominio Web, están íntimamente relacionados con el modelo del dominio de la aplicación. Los modelos OOH que modelan la interfaz Web no pueden existir aisladamente del modelo del dominio, en este sentido, se puede afirmar que los elementos que componen los modelos OOH son primitivas de acceso al modelo del dominio, y por tanto, OOH no promueve la arquitectura multicapa ni el uso de sus patrones arquitectónicos y de diseño asociados. E-WAE ha sido desarrollada con el concepto de la arquitectura multicapa en mente, centrándose solo en la capa de presentación Web, la cual puede ser diseñada y generada de forma desacoplada del resto de capas. Con E-WAE los componentes de la capa de presentación Web pueden relacionarse con los componentes de otras capas de la aplicación siguiendo los catálogos de patrones arquitectónicos y de diseño, que en su mayoría están descritos usando UML.

A partir de los modelos de interfaz Web propuestos por OOH es posible generar, siguiendo la filosofía MDD, una interfaz Web por defecto que puede considerarse como un prototipo de aplicación. Sin embargo en OOH el desarrollo del prototipo requiere la construcción del modelo del dominio. Por el contrario, E-WAE permite el desarrollo de prototipos de aplicaciones, llamados mockups navegacionales, desde los modelos que representan la capa de presentación Web. Por tanto, los prototipos generados no requieren la ejecución de lógica de negocio ni acceso a datos. De esta forma, con E-WAE es posible alcanzar el desacoplamiento de capas que componen una aplicación, como la arquitectura multicapa lo promueve.

Finalmente, tanto OOH como E-WAE tienen en cuenta los roles de usuarios presentes en el dominio de una aplicación. OOH sugiere desarrollar un modelo navegacional por cada rol del dominio, mientras E-WAE propone incluir el control de acceso basado en roles (RBAC) en los modelos navegacionales. En este sentido, el enfoque seguido por E-WAE puedes considerarse más pragmático, dado que ha sido desarrollado con la idea de soportar frameworks empresariales que generalmente implementan el control de acceso basado en roles.

2.5 Interaction Flow Modeling Language – IFML

IFML ha sido adoptado como un lenguaje estándar por la OMG (OMG, 2015) para definir modelos PIM que describen el contenido, la interacción de los usuarios y el comportamiento de las interfaces de software de cualquier tipo de plataforma. Así, IFML permite modelar aplicaciones Web, aplicaciones RIA, aplicaciones móviles, aplicaciones de escritorio, o aplicaciones de control embebidas, entre otras.

IFML es la evolución de WebML (Ceri et al., 2002), pero desde el punto de vista de este trabajo, se podría definir como una generalización de WebML a cualquier tipo de plataforma software. La notación y conceptos no varían mucho con respecto a WebML, incluso, las diferencias pueden resumirse en tres puntos (Brambilla & Butti, 2012): (i) IFML no soporta el modelado de presentación que describe el aspecto visual de la interfaz de usuario incluyendo *look & feel*; (ii) IFML define rigurosamente el concepto de evento, incluyendo varios tipos, como eventos lanzados por el usuario y eventos lanzados por otros sistemas; y (iii) IFML elimina la sucesión de componentes que representan ejecución de lógica de negocio, delegando el modelado de estos a otro tipo de diagramas como diagramas UML de actividad. Sin embargo, al igual que WebML, IFML soporta la sucesión de componentes que representan operaciones CRUD.

IFML propone el desarrollo de tres modelos principales para definir una aplicación: el modelo estructural, que define la estructura de datos y su organización; el modelo de composición, que define la estructura de la interfaz de usuario en términos de componentes de vista, páginas y áreas; y el modelo de navegación que define la navegación entre los componentes del modelo de composición. Por la relevancia que tiene para este trabajo en las siguientes secciones se describe los modelos de composición y navegación.

2.5.1 Modelo de composición en IFML

Los componentes de vista (*View Components*) son los elementos más básicos para definir el modelo de composición IFML. Estos pueden asociarse directamente a entidades que representan orígenes de datos, permitiendo la visualización de instancias de esas entidades. IFML define varios tipos de componentes de vista que cubren una gran variedad de componentes de interfaz de usuario utilizados en aplicaciones modernas y permite la definición de nuevos componentes. Entre los componentes de vistas más utilizados están:

- Detalles (*Details*): visualiza información detallada de una instancia de una entidad.
- Lista (*List*): visualiza múltiples instancias de una entidad permitiendo desplazamiento y ordenación dinámica.
- Jerarquía (*Hierarchy*): visualiza múltiples instancias de diferentes entidades relacionadas jerárquicamente, tales como relaciones maestro-detalle.
- Desplazamiento (*Scroller*): visualiza una secuencia de entidades permitiendo el desplazamiento de la interfaz de usuario.
- Formulario (*Form*): visualiza un formulario para entrada de datos relacionados con una entidad.
- Mensaje (*Message*): visualiza un cuadro de mensaje en la interfaz de usuario.

Las páginas (*Pages*) y áreas (*Areas*) son otros tipos de elemento que conforman el modelo de composición y que en IFML se llaman contenedores (*Containers*). Las páginas pueden estar compuestas por uno o más componentes de vista y otras páginas, representando elementos de la interfaz solicitados por la interacción del usuario. Todos los elementos definidos en la misma página se visualizan uno encima del otro y ocupan el ancho máximo

de la página. Las áreas pueden estar compuestas por páginas y otras áreas, representando grupos de contenedores funcionalmente relacionados.

2.5.2 Modelo de navegación en IFML

El modelo de navegación define el flujo navegacional por medio de conexiones dirigidas entre las páginas y compontes de vista del modelo de composición. Las conexiones pueden utilizarse para definir navegación y flujos de información entre los componentes. La navegación se define a través de una conexión de tipo flujo de navegación normal (*Normal Navigation Flow*), representada por flechas continuas y puede implementarse como un enlace que denota una transición en la interfaz de usuario.

Los flujos de información (*Data Flows*) definen dependencias de datos entre los componentes. Se representan por flechas discontinuas y generalmente no tienen implementaciones concretas. Los datos que se mueve en un flujo de información se representan a través de vinculación de parámetros (*Parameter Binding*) que definen un par nombre-valor que usualmente se corresponde con el identificador de una entidad seleccionada en el componente origen. IFML también define la vinculación de parámetros agrupados (*Parameter Binding Group*) para representar el flujo de conjuntos de datos. Estos son útiles cuando el componente origen de la conexión es de tipo lista o sus derivados.

En IFML, el concepto de vista del sitio (*Site View*) se define como un paquete que contiene los modelos que representan la interfaz de usuario de una aplicación completa. Usualmente se compone de grupos de áreas, páginas y componentes de vistas relacionados a través de flujos navegacionales y de información.

Además, IFML permite definir dentro del modelo de navegación, componentes operacionales (*Operational Components*) que representan operaciones CRUD. Estos componentes no tienen implementación en la interfaz de usuario, por tanto se representan gráficamente fuera de los componentes visuales. Cada componente operacional debe tener al menos un flujo de información de entrada y puede tener dos tipos de flujos de salida: *OK Flow* y *KO Flow*, los cuales denotan respectivamente que la ejecución de la operación CRUD tuvo éxito o fracaso.

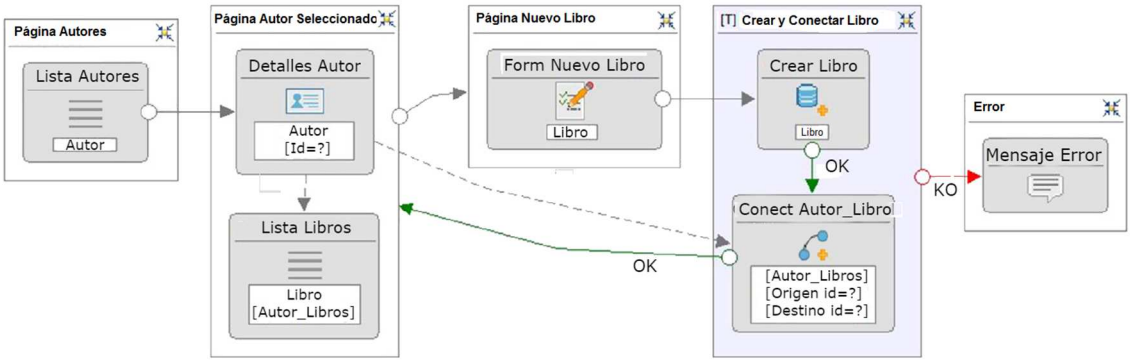
La Figura 2.8 muestra un ejemplo de un modelo de composición-navegación en IFML que modela la navegación desde una entidad *Autor* a entidades *Libro* permitiendo además la creación de los últimos. Nótese la similitud con el modelo WebML discutido en la

sección 2.1. La primera página, *Página Autores*, muestra una lista de autores, el usuario puede seleccionar uno navegando a la página *Página Autor Seleccionado*. Ésta muestra los detalles del autor y una lista de entidades *Libro* relacionadas a éste. El componente *Detalles Autor* tiene una restricción de clave y un flujo navegacional desde *Lista Autores*, indicando que el componente muestra los detalles del autor seleccionado en la lista. El componente *Lista Libros* tiene un flujo de datos desde *Detalles Autor* y una restricción que indica que sólo los libros relacionados con la entidad mostrada en *Detalles Autor* son visualizados.

Desde la página *Página Autor Seleccionado* el usuario puede navegar a la *página Página Nuevo Libro* por medio de un flujo navegacional entre las dos páginas. La página destino contiene un formulario que permite recoger la información para crear una nueva entidad *Libro*, la cual será automáticamente enlazada al autor visualizado en *Página Autor Seleccionado*. Inicialmente, el componente operacional *Crear Libro* crea la nueva instancia de un libro. El flujo de salida *OK Flow* transporta el identificador de la nueva entidad como parámetro vinculante, el cual se usa en la expresión condicional *Origen*. El identificador del autor se obtiene desde el flujo de datos entrante que viene desde la entidad *Detalles Autor*, el cual se corresponde con la expresión condicional *Destino*. Si la ejecución del componente operacional *Conectar Autor_Libro* tiene éxito, el flujo navegacional retorna a la página *Página Autor Seleccionado*.

En la Figura 2.8 se puede notar como IFML soporta la definición de transacciones para grupos de componentes operacionales. Si la ejecución de uno de los componentes operacionales dentro de la transacción falla, el flujo navegacional se dirige a una página con un mensaje de error.

Figura 2.8: Modelo navegacional en IFML.



Al igual que WebML, los modelos IFML están formados por componentes conceptuales distinguidos a través de un icono, pueden estar relacionados a una entidad de una fuente

de datos y pueden expresar restricciones o condiciones sobre las entidades relacionadas. De hecho, el uso de componentes conceptuales unidos a través de enlaces conceptuales definidos con rigurosa semántica permite inferir el flujo de datos entre componentes y en última instancia el comportamiento de una interfaz de usuario. Como se puede ver, esta es la principal característica de WebML que ha sido transferida íntegramente a IFML. Esta similitud hace posible que la plataforma WebRatio sea reutilizada para soportar IFML como su notación oficial. WebRatio ofrece un conjunto de editores de diagramas para desarrollar los modelos IFML y generar automáticamente desde ellos aplicaciones Web y aplicaciones móviles listas para ser ejecutadas (aplicaciones para otras plataformas aún no son soportadas). Estas características hacen que IFML comparta todas las ventajas y desventajas de WebML discutidas en la sección 2.1.

2.5.3 IFML y UML

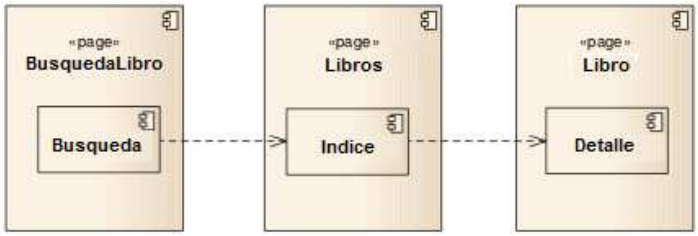
La especificación formal de IFML define el meta-modelo IFML, el cual específica la estructura y semántica de sus constructores usando MOF, y el perfil UML para IFML, que define una sintaxis basada en UML para expresar los modelos IFML (OMG, 2015).

El meta-modelo MOF que define IFML extiende el meta-modelo de UML. Por medio de la definición de éste meta-modelo, IFML puede considerarse como un lenguaje de dominio específico (*Domain Specific Language*, DSL). La sintaxis y semántica de los elementos del lenguaje son definidos para representar exactamente las características de las interfaces de usuario. La desventaja de este enfoque es que las herramientas UML CASE genéricas no pueden soportar el nuevo lenguaje, incluyendo la edición de los modelos basados en el meta-modelo (por ejemplo, el diagrama de la Figura 2.8 no puede ser editado en una herramienta UML CASE genérica).

El perfil UML para IFML permite que, en teoría, la notación pueda ser administrada por herramientas UML CASE genéricas. Sin embargo, su origen basado en WebML, hace que en la práctica sea una notación no UML. El alto número de estereotipos que define, más de 75, y el hecho de que la mayoría de ellos extiendan elementos de los diagramas de componentes, hace imposible, en la práctica, proporcionar diagramas UML de clase y UML de secuencia para modelar la capa de presentación de aplicaciones Web. La Figura 2.9 muestra un diagrama UML extendido con el perfil IFML que modela una funcionalidad simple y habitual en las interfaces de usuario: una vista que permite buscar entidades a partir de una palabra clave y la navegación a una vista que muestra los resultados de la búsqueda. Como puede verse, a pesar la sencillez de la funcionalidad, el

diagrama está compuesto por elementos UML de componentes, los cuales no pueden integrarse directamente con elementos cercanos al nivel de implementación. Por tanto la inclusión de patrones arquitectónicos y de diseño es muy difícil, si no imposible en IFML. Además, la naturaleza abstracta de los estereotipos que define no permite que estos sean traducibles directamente a código orientado a objetos, por lo cual IFML requiere de herramientas propietarias para generar código a partir de ellos (*black-box wizards*). Por tanto, sin estos generadores de código, IFML se centra en la producción de diseños de representación en vez de asistir al proceso de diseño, un problema familiar en casi todos los enfoques orientados al modelado (Barry & Lang, 2001).

Figura 2.9: Modelo UML extendido con el perfil UML para IFML.



2.5.4 IFML y E-WAE

Como se ha comentado, IFML es la evolución de WebML que mantiene y generaliza sus principales características a otros tipos de plataformas software. Además, dado que WebRatio también ha evolucionado para soportar IFML, pero cumpliendo el mismo rol que en la metodología WebML (generador de código desde modelos PIM), la discusión desarrollada en la sección 2.1.3 puede ser aplicada también para el caso de IFML.

Una de las principales ventajas de IFML es su estandarización por OMG y la definición formal del lenguaje a través del meta-modelo IFML que extiende el meta-modelo UML. Esta definición formal asegura, al menos en teoría, el desarrollo de reglas de transformación que puedan traducir los modelos PIM IFML en modelos PSM y finalmente a código de interfaz de usuario para cualquier plataforma.

Sin embargo, este proceso puede llegar a ser muy complejo dado que requiere del conocimiento profundo del meta-modelo de IFML y de la definición de un meta-modelo para los modelos PSM (que puede o no ser el mismo meta-modelo de UML) para permitir desarrollar las reglas de transformación a nivel de meta-modelos. Al contrario, dado que tanto E-WAE como WAE4x están basados en el meta-modelo de UML, comparten el

mismo núcleo semántico, y por tanto es fácil definir e implementar las reglas de transformación (OMG, 2013).

Como se comentó, el perfil UML para IFML permite desarrollar los diagramas IFML con herramientas UML CASE genéricas. Sin embargo, el alto número de estereotipos que define el perfil (más de 75) y su naturaleza abstracta hace que no sea posible obtener código a partir de ellos. Por tanto IFML se centra en la producción de representaciones del diseño sin asistir directamente al proceso de desarrollo de software. Para obtener código, IFML al igual que WebML, requiere de WebRatio, el cual vincula el mantenimiento del código a una herramienta propietaria.

Finalmente, IFML se centra en la especificación de interfaces de usuario de cualquier tipo de plataforma, E-WAE se centra en la especificación de la capa de presentación de aplicaciones Web empresariales. Tanto IFML como E-WAE aíslan la especificación del *front-end* de los detalles de implementación facilitando la comunicación entre desarrolladores y usuarios finales.

2.6 Desarrollo Dirigido por Mockups – MockupDD

Como se ha comentado en las secciones anteriores, las metodologías de Ingeniería Web dirigidas por modelos (MDWE) como WebML y UWE se basan en modelos de alto nivel de abstracción que especifican las aplicaciones Web y actúan como modelos fuente a partir de los cuales se genera el código de las aplicaciones por medio de generadores de código propietarios. Estas metodologías comparten un proceso de desarrollo común que puede resumirse en:

1. Generación del modelo de datos/dominio. Define las entidades de datos/dominio y sus relaciones subyacentes a la aplicación.
2. Generación de modelo de navegación/hipertexto. Define los nodos navegacionales y sus enlaces especificando los datos que serán publicados y como serán alcanzados en el camino navegacional.
3. Generación del modelo de presentación. Define las características de la interfaz de usuario de los nodos navegacionales.

Generalmente, estos modelos se generan por medio de transformaciones (semi)automáticas que traducen y refinan el modelo anterior generando en última instancia las aplicaciones Web ejecutables. Por tanto, estas metodologías centradas en el

refinamiento de modelos no mencionan a los usuarios finales en el proceso desarrollo de las aplicaciones.

Al contrario, los procesos ágiles, para intentar asegurar que las aplicaciones cumplen con los requisitos y se ajustan adecuadamente a los cambios en estos requisitos, promueven la interacción constante con los usuarios finales por medio de la entrega y validación de prototipos en cortos periodos de tiempo. Los procesos ágiles argumentan que las especificaciones del software deben emerger naturalmente mejorando los prototipos junto con el desarrollo hasta que se obtiene la aplicación final (Stober & Hansmann, 2010).

Por tanto, mientras las metodologías de Ingeniería Web facilitan la especificación de las aplicaciones, la portabilidad, abstracción y productividad, no promueven la interacción con los usuarios finales durante el proceso de desarrollo. Esta característica la proporcionan los procesos ágiles. Sin embargo, estos se basan en la implementación directa de las aplicaciones, y por tanto, no proporcionan abstracción, portabilidad y productividad a través de la generación automática de código.

Rivero et al. (2011) y (2014) proponen una metodología ágil basada en modelos. Esta metodología llamada Desarrollo Dirigido por Mockups (*Mockup-Driven Development - MockupDD*) intenta llevar los beneficios de los procesos ágiles a las metodologías de la ingeniería Web.

La metodología promueve la rápida construcción de mockups de interfaz de usuario, lo más simple posible, en pequeños incrementos y permitiendo la participación activa de los usuarios finales. Los mockups se procesan para obtener modelos que representan la estructura de la interfaz de usuario (*Structural User Interface Model - SUI*), los cuáles se enriquecen con etiquetas que la metodología define y actúan como modelos fuente para generar los modelos de presentación abstracta de las metodologías de Ingeniería Web. Por medio de heurísticas que la metodología especifica, y con base en los modelos SUI, los modelos de presentación abstracta y los requisitos navegacionales, se obtienen los otros modelos abstractos de las metodologías de Ingeniería Web (modelos navegacionales y de datos/dominio).

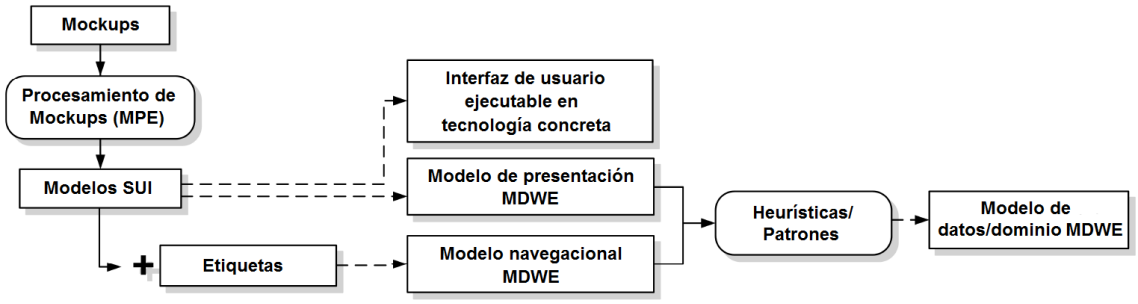
Por tanto, la metodología está dirigida por la activa participación de los usuarios finales y está asistida por el uso de modelos como las metodologías de la Ingeniería Web promueven. En resumen, la metodología: (i) soporta la inclusión de los usuarios finales en el proceso de desarrollo desde las primeras etapas; (ii) soporta la generación continua de prototipos; (iii) promueve el desarrollo iterativo, facilitando la introducción de nuevas

características en entregas cortas; y (iv) se integra con metodologías de ingeniería Web por medio de transformaciones de modelos.

2.6.1 El proceso de desarrollo con MockupDD

La Figura 2.10 muestra el proceso de desarrollo con MockupDD. El proceso comienza con la construcción de los mockups de interfaz de usuario por medio de una herramienta propia propuesta por la metodología (aunque la metodología también soporta algunas herramientas industriales como Balsamiq[5] y Pencil[6]). Usualmente los mockups se definen junto con otras especificaciones como casos de uso, historias de usuario o anotaciones informales.

Figura 2.10: Proceso de desarrollo con MockupDD



El siguiente paso es transformar los mockups en modelos que representan la estructura de la interfaz de usuario (modelos SUI). Este paso comprende el procesamiento de los ficheros producidos por la herramienta generadora de mockups a través de una serie de procesadores que conforman un componente clave de la metodología llamado MPE (*Mockup Processing Engine*). Como resultado del procesamiento, se obtienen los modelos SUI.

Dado que la mayoría de los meta-modelos de presentación propuestos por las metodologías de ingeniería Web comparten los mismos conceptos con el meta-modelo que define los modelos SUI (páginas, enlaces, botones, cajas de texto, etc.), es posible obtener los modelos de presentación abstracta de las metodologías de ingeniería Web a partir de los modelos SUI a través de una simple traducción.

[5] https://balsamiq.com/

[6] http://pencil.evolus.vn/

Mientras la mayoría de las metodologías de ingeniería Web no permiten la generación de código directamente desde los modelos de presentación, MockupDD proporciona una herramienta generadora de código desde los modelos SUI que facilita la generación de prototipos ejecutables que pueden ser usados para validar los requisitos con los usuarios finales. En consecuencia, como muestra la Figura 2.10, MockupDD permite, a partir de los modelos SUI, generar modelos conformes a las metodologías de ingeniería Web o seguir una metodología basada en código en la cual los prototipos ejecutables se generan y enriquen incrementalmente.

Los modelos SUI describen la estructura de la interfaz de usuario sin tener en cuenta su comportamiento ni sus datos subyacentes. Por tanto, para obtener los modelos navegacionales de las metodologías de ingeniería Web, MockupDD define un conjunto de etiquetas que pueden aplicarse iterativamente sobre los elementos de los modelos SUI, proporcionando sugerencias para producir los modelos navegacionales. Por ejemplo, dos de las etiquetas más comunes para producir modelos navegacionales son *nodo* y *enlace*. La primera se aplica sobre los elementos página de los modelos SUI asignándole un identificador único para propósito navegacionales. La segunda se aplica a los elementos capaces de lanzar procesos navegacionales, como los botones, asignándoles un identificador que representa la página destino. Usando esta información proporcionada por las etiquetas, es posible obtener los modelos navegacionales de las metodologías de Ingeniería Web.

Después de obtener los modelos de presentación y navegación a través de una traducción directa y por medio de las etiquetas respectivamente, es posible obtener el modelo de datos/dominio a través de la aplicación de heurísticas, las cuales han sido diseñadas de acuerdo a los patrones recurrentes encontrados en cada metodología. La aplicación automática de estas heurísticas a través de la herramienta proporcionada por MockupDD suele generar modelos de datos/dominio aproximados, que son efectivos en la mayoría de los casos, pero requieren del ajuste manual por parte de los desarrolladores para alcanzar una versión definitiva de los modelos.

2.6.2 MockupDD y E-WAE

Tanto MockupDD como E-WAE proponen el desarrollo de mockups interactivos como herramientas que pueden ser usadas en la etapa de obtención y formulación de requisitos, debido a que permiten una comunicación sin ambigüedad entre desarrolladores y usuarios finales. En MockupDD, los mockups interactivos se generan automáticamente a partir de

diagramas que representan la estructura de la interfaz de usuario, mientras que en E-WAE los mockups interactivos se generan a partir de modelos bien definidos. En cualquier caso, al igual que los procesos ágiles, los dos enfoques permiten interacciones continuas e informales a través de los mockups con los usuarios finales. Por tanto, pueden ser adecuadas en el desarrollo de aplicaciones en las que los requisitos no están bien definidos, son difíciles de especificar o pueden cambiar rápidamente, incluso durante el desarrollo.

Los dos enfoques han sido desarrollados con la idea de reutilizar los mockups en el proceso de desarrollo de las aplicaciones finales. Por tanto, en los dos enfoques los mockups se relacionan con modelos bien definidos. En MockupDD, los mockups actúan como diagramas fuente a partir de los cuales se generan modelos bien definidos, que, después de una serie de transformaciones, se convierten en modelos especificados por metodologías de Ingeniería Web. En E-WAE, los mockups se generan a partir de modelos bien definidos por medio de reglas de transformación que pueden aplicarse manual o automáticamente.

Los mockups propuestos por MockupDD describen la estructura de la interfaz de usuario, mientras que los mockups propuestos por E-WAE son mockups navegacionales que se centran en los elementos principales de la interfaz de usuario y sus relaciones navegacionales. En MockupDD, los mockups se centran en describir el aspecto visual de interfaz de usuario sin tener en cuenta la arquitectura de las aplicaciones, mientras en E-WAE, los mockups se implementan sobre frameworks industriales que promueven la arquitectura multicapa.

Ésta es una de las diferencias más importantes entre los dos enfoques: MockupDD ha sido desarrollada con la idea de integrarse con las metodologías de Ingeniería Web y por tanto compartir sus beneficios, pero también sus limitaciones. Como se comentó en las secciones anteriores, una de las principales limitaciones de las metodologías de Ingeniería Web es que definen elementos de modelado de alto nivel de abstracción que no permiten incorporar todo el poder de los patrones arquitectónicos y de diseño en los modelos.

Por otro lado, E-WAE, puede ser visto como un intento de reconciliar el diseño de la arquitectura y los procesos ágiles. Desarrolladores y usuarios finales pueden trabajar juntos para descubrir requisitos funcionales a través de los mockups interactivos generados a partir de mapas navegacionales PSM. Debido a que los mapas navegacionales PSM son modelos UML, pueden ser utilizados como documentación bien definida y a la

vez, pueden ser fácilmente extendidos con modelos que incluyan patrones arquitectónicos y de diseño generando código que implementan una arquitectura definida por los desarrolladores. Además, los mapas navegacionales PSMs incluyen componentes Web implementados en frameworks empresariales. Por tanto, los mockups generados a partir de ellos se implementan sobre estos frameworks, permitiendo su reutilización completa en el desarrollo de las aplicaciones finales.

Otra diferencia importante entre MockupDD y E-WAE son las herramientas asociadas. Para añadir agilidad a los procesos de desarrollo, es fundamental el soporte de herramientas que permitan las transformaciones entre modelos y la generación automática de código. MockupDD hace un uso intensivo de herramientas propietarias que pueden ser integradas en Eclipse o Visual Studio soportando la definición de los mockups, su procesamiento para obtener los modelos SUI, la manipulación de estos modelos, la gestión de etiquetas y finalmente la ejecución de los modelos SUI que constituyen los mockups interactivos.

Los modelos propuestos por E-WAE cumplen el estándar UML, por tanto, pueden desarrollarse y mantenerse en cualquier herramienta UML CASE genérica, y la transformación entre modelos y generación de código se especifican de forma explícita. Así, se puede afimar que E-WAE, a través de la especificación de las reglas de transformación bien definidas, promuve su implementación automática. De hecho, para obtener los beneficios de los procesos ágiles, se requiere implementar estas reglas de transformación en herramientas que permita su ejecución automática. El presente trabajo ha implementado las reglas de transformación PIM2PSM conforme al estándar QVT (*Query/View/Transformation*) (OMG, 2008) en la herramienta CASE Borland Together 2008 y las reglas de transformación PSM2Code a través de un conjunto de reglas XSLT compatibles sólo con Borland Together 2008.

2.7 Plataformas de desarrollo visual: Mendix y Outsystems

Las plataformas de desarrollo visual como Mendix y Outsystems son herramientas de alta productividad usadas para desarrollar aplicaciones Web. Estas plataformas ofrecen un entorno de desarrollo integrado donde los desarrolladores generan diagramas visuales que pueden ser directamente desplegados en servidores Web preconfigurados que soportan la plataforma. Aunque estas plataformas no se centran en el modelado de aplicaciones ni en la generación de mockups, se analizan en este trabajo porque proporcionan una especie de pseudo-notación visual para desarrollar aplicaciones Web.

Aunque Mendix y Outsystems comparten la misma filosofía de desarrollo, existen diferencias en la implementación, por ejemplo, Outsystems soporta las plataformas Java y .NET e incluye una fase de compilación de los modelos donde se genera el código de servidor antes de ser desplegado, mientras Mendix solo soporta Java y los modelos son directamente desplegados en el servidor Web, el cual es responsable de interpretar los modelos y generar el código de cliente a partir de ellos. En cualquier caso ninguna de las dos plataformas permite controlar la forma en la que se genera el código a partir de los diagramas.

Este tipo de plataformas promueven el desarrollo por medio de *Lenguajes Visuales Específicos del Dominio* (DSML) propietarios que definen elementos gráficos por medio de los cuales se desarrolla un conjunto predefinido de diagramas que definen el modelo de datos, las interfaces de usuario y el comportamiento de las aplicaciones. De esta forma las plataformas abstraen el desarrollo de una aplicación de los detalles de implementación.

Mendix y Outsystems comparten un proceso de desarrollo común, el primer paso es desarrollar los diagramas que cada plataforma define en su entorno de desarrollo (*Mendix Business Modeler* en Mendix y *Service Studio* en Outsystems). Las dos plataformas definen los mismos tipos de diagramas: el modelo de datos ("*Meta-model*" en Mendix y *Entity Model* en Outsystems), las interfaces de usuario (*Form-models* en Mendix, *Web Screens* en Outsystems) y los diagramas que definen el comportamiento de la aplicación (*Microflows* en Mendix y *Actions* en Outsystems).

El segundo paso es desplegar la aplicación en los servidores Web (*Mendix Business Server* en Mendix y *Platform Server* en Outsystems). Mendix permite desplegar directamente los modelos en el servidor, de forma que cuando se recibe una petición del cliente Web, el servidor interpreta los modelos y genera el código HTML y JavaScript que se envía al cliente. En Outsystems, los modelos se compilan generando el código de servidor (Java o .NET), el cual se despliega en el servidor Web. Como se ha comentado, salvando estas diferencias de implementación, la filosofía de desarrollo de este tipo de plataformas es similar. A modo ilustrativo, a continuación se detalla el proceso de desarrollo con Mendix.

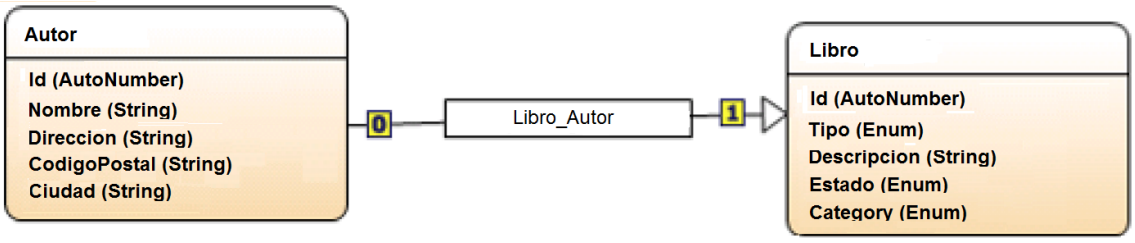
2.7.1 El proceso de desarrollo con Mendix

Mendix soporta el desarrollo de aplicaciones Web a través de tres tipos de diagramas que definen la estructura y comportamiento de las aplicaciones:

- "*Meta-model*": describe la estructura de la aplicación usando una notación propietaria de Mendix.
- *Form-models*: describen la interfaz de usuario de la aplicación a través de formularios, los cuales suelen estar compuestos de controles de usuario estándar dispuestos siguiendo la filosofía *drag & drop*.
- *Microflows*: describen la ejecución de lógica de negocio a través de una notación propietaria de Mendix similar a los diagramas de actividad UML.

Nótese que el nombre "Meta-Model" del diagrama que describe la estructura de la aplicación puede ser inapropiado, debido a que el tipo de modelos a los que se refiere Mendix, son modelos simples, que de acuerdo a (OMG, 2003) se definen en el nivel M1 y no en el nivel M2 donde están definidos los verdaderos meta-modelos. La Figura 2.11 muestra un ejemplo de un "Meta-model" en Mendix que describe las entidades *Autor* y *Libro* y la relación entre ellas.

Figura 2.11: Diagrama "Meta-modelo" en Mendix

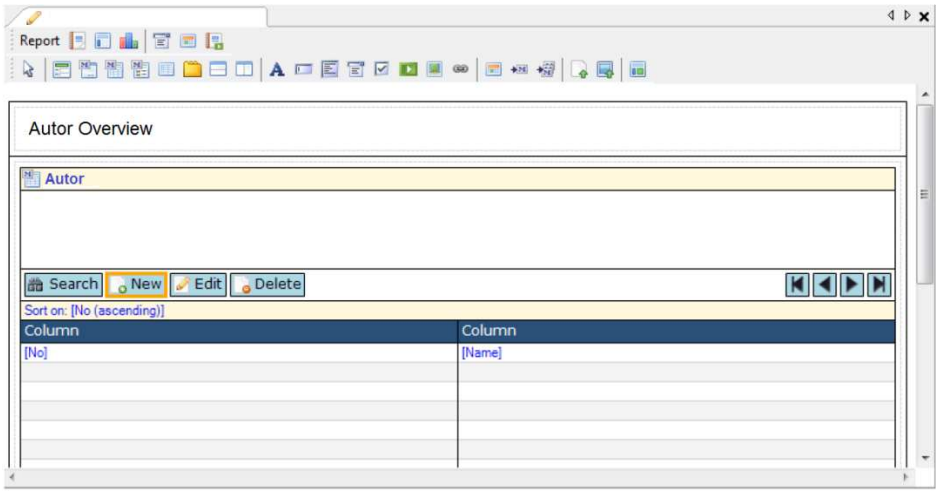


Como puede observarse en la Figura 2.11, Mendix usa su propia notación para describir las asociaciones entre entidades. Esta asociación debe ser interpretada como "un libro se relaciona exactamente a un autor", "un autor puede tener cero libros relacionados", la flecha sin rellenar indica que en la asociación, la entidad *Autor* actúa como entidad maestra y la entidad *Libro* como entidad detalle. Este ejemplo, que puede no tener sentido en el dominio real, ilustra claramente como la notación definida por Mendix puede ser de difícil lectura para los usuarios familiarizados con UML; la lectura de las multiplicidades es en el sentido inverso y la flecha sin rellenar tiene una semántica muy diferente.

Las entidades definidas en el "Meta-model" pueden relacionarse con los otros tipos de diagramas. Es posible lanzar la ejecución de lógica de negocio especificada en un Microflow por medio de la definición de eventos en las entidades, por ejemplo, es posible lanzar un Microflow cuando una entidad es actualizada o eliminada.

Para definir la interfaz de usuario, Mendix proporciona un editor de formularios similar a los entornos de desarrollo proporcionados por Eclipse o Visual Studio. Mendix pone a disposición de los desarrolladores los controles de usuario estándar que pueden incrustarse en los formularios siguiendo la filosofía *drag & drop*. Los formularios pueden relacionarse con el "Meta-model" para permitir mostrar y persistir la información. La Figura 2.12 muestra un formulario relacionado a la entidad *Autor* mostrada en la Figura 2.11. Cuando un formulario se relaciona con una entidad del "Meta-model", Mendix proporciona automáticamente la implementación de las operaciones CRUD básicas mostradas en la Figura.

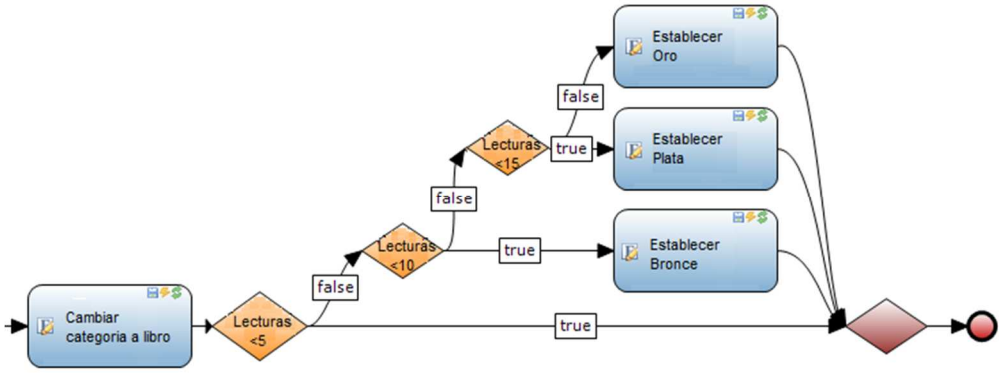
Figura 2.12: Formulario para la entidad *Autor* en Mendix



Al igual que el "Meta-model", los formularios pueden definir eventos para lanzar Microflows. Por ejemplo es posible agregar un botón a un formulario y conectarlo a un Microflow. Si éste modifica una entidad mostrada por el formulario, el cambio se refleja instantáneamente en el formulario sin escribir una sola línea de código.

Para definir la lógica de negocio de la aplicación, Mendix ofrece los Microflows, los cuales pueden ser lanzados por eventos desde los formularios o desde el "Meta-model". A través de los Microflows, los desarrolladores pueden modificar entidades, controlar cómo y cuándo se muestran los formularios e integrarse con sistemas externos usando servicios Web. La Figura 2.13 muestra un Microflow que permite establecer la categoría de un *Libro* de acuerdo al número de lecturas que ha tenido. El editor de Microflows permite expresar lógicas de aplicación mucho más complejas, incluyendo bucles, llamadas a otros Microflows, aperturas de formularios, llamadas a servicios Web externos, ejecuciones de código Java, etc.

Figura 2.13: Diagrama de Microflow en Mendix



Finalmente, los diagramas tienen que ser desplegados en el servidor. En esta etapa, los diagramas son transferidos al repositorio de modelos (*Mendix Model Repository*) y se crea automáticamente una base de datos basada en el "Meta-modelo". Cuando se inicializa el servidor Web de Mendix (*Mendix Business Server*) y se recibe una petición Web, el servidor interpreta los modelos y genera el código listo para ser enviado al cliente Web.

2.7.2 Las plataformas de desarrollo visual y E-WAE

La principal diferencia entre E-WAE y las plataformas de desarrollo visual es que estas no son compatibles con la filosofía de desarrollo MDD. En este sentido, se podría afirmar que estas plataformas trabajan en un solo nivel de abstracción, y que este único nivel de abstracción es el más adecuado para aplicaciones simples, porque abstrae la arquitectura, la inclusión explicita de patrones de diseño, el diseño del código, la integración con la base de datos, etc.

Para aplicaciones que requieran características más allá de las soportadas, las plataformas ofrecen la posibilidad de ser extendidas por medio de código, lo cual implica reducir el nivel de abstracción requiriendo de amplios conocimientos, tanto de la plataforma como de desarrollo de código, y, por tanto reduciendo la simplicidad de desarrollo, su principal beneficio. De igual forma agregar modelos de más nivel de abstracción que se integren en el proceso de desarrollo podría llegar a ser muy difícil, si no imposible, debido a que los diagramas visuales no están formalmente definidos a través de meta-modelos o notaciones bien definidas.

Por el contrario, E-WAE ha sido desarrollada con la filosofía de MDD en mente. Define meta-modelos que proporcionan especificaciones precisas de los elementos definidos en los modelos PIM y PSM, y un conjunto de reglas bien definidas que permite la

transformación manual y automática de los modelos PIM en PSM, y estos en código. Estas reglas bien definidas facilitan la trazabilidad dentro del proceso MDD, precisando los diseños en los cuales el código está basado, dando la flexibilidad de ajustarlas de acuerdo a las necesidades de las aplicaciones e independizando la generación y mantenimiento de código de herramientas específicas.

Generalmente, los diagramas propuestos por estas plataformas son fáciles de entender. No obstante, se requiere previo conocimiento de su notación. De hecho una desventaja importante de estas plataformas es el uso de notaciones propietarias que pueden ser modificadas sin previo aviso y en algunos casos, aunque puedan tener una notación gráfica similar a otras notaciones pueden tener una semántica totalmente diferente. Por ejemplo, las asociaciones en los diagramas "Meta-modelo" de Mendix pueden generar importantes errores de lectura para usuarios familiarizados con diagramas UML de clases. Por el contrario los modelos propuestos por E-WAE son modelos UML, el estándar *de facto* reconocido tanto en la academia como en la industria.

Se podría afirmar que los modelos propuestos por estas plataformas son totalmente ejecutables debido a que no requieren de código por parte de los desarrolladores para generar aplicaciones completamente funcionales. Sin embargo, estas plataformas no dan la flexibilidad de modificar la forma en la que se interpretan los modelos (en el caso de Mendix) o la forma en la que se genera el código (en el caso de Outsystems). En este sentido, las plataformas actúan como generadores de código propietarios (*black-box wizards*) que ocultan la generación de código desde sus modelos con un conjunto predefinido de elementos. Por el contrario, E-WAE proporciona un conjunto de reglas de transformación bien definidas que promueven su implementación automática permitiendo la flexibilidad de ser ajustadas de acuerdo a los requisitos propios de las aplicaciones.

Como se ha comentado, la principal fortaleza de las plataformas de desarrollo visual es que permiten el desarrollo inmediato de aplicaciones simples sin necesidad de escribir código. Un análisis de este tipo de plataformas de desarrollo visual (Rymer et al., 2005) asegura que estas buscan desplazar a las plataformas de desarrollo de código como Java o .NET, pero que no ofrecen toda la flexibilidad requerida para el desarrollo de aplicaciones de mediana complejidad. El análisis concluye que se puede adoptar estas plataformas para disminuir el coste del desarrollo de software pero en detrimento del diseño arquitectónico y el uso de estándares. Por el contrario, E-WAE promueve el desarrollo de software basado en el diseño arquitectónico y en el modelado. Respecto a la arquitectura, promueve el desarrollo basado en la arquitectura multicapa, y respecto al

modelado, utiliza el estándar UML. Respecto a la generación de código, E-WAE es totalmente compatible con las plataformas de desarrollo de código Java y .NET. De hecho, E-WAE promueve la codificación y mantenimiento de las aplicaciones en un IDE por desarrolladores. Así, el código generado está listo para ser importado en Eclipse (en el caso de Java) o Visual Studio (en el caso .NET) y ser complementado por los desarrolladores.

Las plataformas de desarrollo visual no tienen en cuenta las metodologías de ingeniería del software que soportan el diseño y mantenimiento de las aplicaciones, los diagramas que proponen se centran en el desarrollo rápido de las aplicaciones, pero no en su mantenimiento. De hecho, el modelo de desarrollo de Mendix tiene las siguientes etapas (Mendix, 2016b): diseño, construcción, despliegue, administración, iteración y colaboración. Las trece reglas de desarrollo definidas por Outsystems son (Mendes, 2016): validar el impacto, provisión inmediata, anticipación al fallo, desarrollo basado en la producción, integración continua, documentación de las modificaciones, registro de configuraciones, orquestar el proceso de despliegue, ciclos de entrega compactos, medición de todos los aspectos, identificaciones de objetivos y desviaciones, ampliar bucles de retroalimentación. Por tanto, ni Mendix ni Outsystems mencionan el mantenimiento de las aplicaciones en sus etapas y reglas respectivamente. Como el mantenimiento no está dentro de sus objetivos, estas plataformas no proporcionan diseños bien definidos que permitan incluir patrones arquitectónicos y de diseño. Además, los diagramas proporcionados no son representaciones directas de código, requiriendo el generador de código automático. Por tanto, estas plataformas no promueven la codificación por parte de los desarrolladores y acoplan al mantenimiento de las aplicaciones a sus motores de generación de código.

Debido a que E-WAE proporciona modelos PSM basados en UML que pueden ser directamente transformados en código, el mantenimiento de este no depende de ninguna herramienta en particular. Esta es una característica muy importante que ya ha sido reconocida en el sector industrial. Natis et al. (2015) presentan un análisis que critica a Outsystems por no permitir a los desarrolladores tener un control sobre el código generado. Por ejemplo, si se modifica el código generado por la plataforma, estas modificaciones no pueden ser llevadas a los modelos, convirtiéndose estos en modelos obsoletos. Ésta es una limitación presente en todas las plataformas que no proporcionan modelos PSM basados en UML.

2.8 Web Application Extension – UML-WAE

Una de las características más relevantes de la notación UML es su capacidad para definir especializaciones del lenguaje. Conallen (1999) presenta una especialización de la notación UML denominada UML-WAE que permite reutilizar toda la notación UML para modelar componentes del dominio Web.

Conallen diferencia entre un sitio y una aplicación Web. Un sitio Web es básicamente estático, aunque que puede incluir simples operaciones CRUD. Sin embargo, una aplicación Web es fuertemente dinámica y dispone de un conjunto de reglas de negocio que pueden reaccionar y alterar el estado de la aplicación a partir de la interacción con los usuarios u otros sistemas extenos. Su desafío es la complejidad, ya que requiere implementar una arquitectura que se adapte a los cambios constantes, que facilite su ágil integración con otros sistemas y que resuelva picos variables de interacción con un buen rendimiento. Para afrontar este desafío los autores proponen modelar con UML todos los componentes que forman parte de una aplicación Web.

Sin embargo, cuando se intenta modelar aplicaciones Web con UML se encuentra que algunas de sus características no encajan dentro de ningún elemento de modelado de UML estándar. Por ejemplo, estas aplicaciones incluyen páginas y enlaces (*links*), los cuales no se pueden caracterizar con conceptos del diseño orientado a objetos. Además UML estándar no facilita la tarea de diferenciar entre componentes que se ejecutan en el lado del servidor y componentes que se ejecutan del lado cliente. Con el fin de conseguir una única notación para todo el sistema (componentes Web y componentes tradicionales de las capas internas) UML-WAE extiende a UML usando su propio mecanismo de extensión formal: los perfiles UML.

UML-WAE es un perfil UML que extiende su sintaxis y su semántica para expresar los conceptos específicos del dominio Web. UML-WAE ha sido diseñado de tal forma que los componentes específicos del dominio Web pueden integrarse con el resto de modelos UML del sistema exhibiendo el nivel de abstracción y detalle adecuado para diseñadores y desarrolladores de aplicaciones Web. Por tanto, UML-WAE permite aprovechar eficientemente UML como lenguaje de modelado para desarrollar aplicaciones Web, caracterizar adecuadamente una aplicación Web con una sola notación estándar, manteniendo su trazabilidad e integridad, y finalmente, incluir explícitamente todo el catálogo de patrones arquitectónicos y de diseño definidos en UML y que han sido probados en la industria durante años. De esta forma UML-WAE permite reutilizar todos

los beneficios de UML, incluyendo el conocimiento, experiencia y herramientas asociadas a este en el desarrollo de aplicaciones Web.

2.8.1 El proceso de desarrollo con UML-WAE

El proceso de desarrollo con UML-WAE asume que las aplicaciones Web son aplicaciones centradas en lógica de negocio, por tanto los modelos más importantes del sistema deben hacer énfasis en la lógica y el estado del negocio y no en la capa de presentación. Si las características de presentación son importantes o de alta complejidad deberían modelarse independientemente del modelo de negocio.

UML es el estándar más utilizado para especificar y documentar sistemas intensivos en software. Así, UML-WAE se utiliza para modelar las características propias del dominio Web que no pueden ser modeladas con UML estándar siguiendo su misma filosofía: el número de nuevos estereotipos que define es mínimo para caracterizar los componentes del dominio Web, la semántica de estos nuevos estereotipos se define en términos de código, y los modelos se basan en el uso de diagramas UML de clases y de secuencia como define el Proceso de Desarrollo Unificado (Jacobson et al., 1999).

En el proceso de desarrollo con UML-WAE, la lógica de negocio ejecutada detrás del servidor Web, se modela a través de los diagramas de UML estándar (diagrama de casos de uso, diagramas de clase, diagramas de secuencia, diagramas de componentes, etc.). El perfil UML-WAE se utilizada para modelar los componentes específicos de los entornos Web y la integración con el resto de la aplicación. En concreto, a través del perfil UML-WAE se modela las páginas Web, los enlaces entre páginas Web y el contenido dinámico generado por el servidor y presentado por el cliente Web. De esta forma, en una arquitectura multicapa, UML-WAE se utiliza para modelar la capa de presentación Web.

Para modelar los componentes propios del dominio Web de forma consistente con el resto de modelos UML, estos se traducen a elementos de UML estándar a los cuales se les aplica adecuadamente los estereotipos, valores etiquetados y restricciones definidos en el perfil UML-WAE. Por ejemplo, las páginas Web son traducidas a clases UML a las cuales se les aplica los estereotipos *ClientPage* si la página está en el lado del cliente, es decir solo contiene código HTML o XML, o *ServerPage* si la página contiene código que será ejecutado por el servidor mientras está siendo preparada. Los enlaces entre páginas representan caminos navegacionales desde una página a otra, por tanto son traducidos a asociaciones UML entre páginas con estereotipos *Link, Build, Submit*, etc.

El proceso de modelado con UML-WAE hace una clara distinción entre la estructura de navegación y la apariencia de la interfaz de usuario. UML-WAE sólo se ocupa de modelar la navegación por la aplicación. Conceptos de la presentación como componentes de interfaz de usuario no se modelan más allá de la división de la ventana en regiones. Si por necesidades de la aplicación es necesario modelar la interfaz de usuario, éste se debe construir como un modelo independiente del modelo construido con UML-WAE.

En el modelado de una aplicación Web con UML-WAE es necesario describir todas las páginas Web y sus relaciones. En UML-WAE una página Web es cualquier elemento que puede ser generado por un servidor Web. Como se comentó anteriormente, en UML-WAE las páginas Web se representan por clases UML estereotipadas y los enlaces entre ellas como asociaciones estereotipadas entre clases. En UML-WAE cada página Web que es construida dinámicamente se modela a través de dos elementos: una clase estereotipada con *ServerPage* y una clase estereotipada con *ClientPage*. La clase *ServerPage* representa la página conceptual que se ejecuta del lado del servidor cuando un cliente Web hace una petición, y la clase *ClientPage* representa la página vista por el cliente Web, en esencia constituye la interfaz de usuario de la aplicación que ha sido creada dinámicamente por la clase *ServerPage*.

Como es lógico la clase *ServerPage* tiene acceso a los recursos del lado del servidor (componentes en las capas intermedias de la aplicación). Sus métodos pueden ser vistos como scripts que se ejecutan del lado del servidor y sus atributos como variables en el ámbito de la página que pueden ser modificados por la lógica que se ejecuta en las capas intermedias de la aplicación. En este sentido, las clases *ServerPage* son clases que participan en la lógica de negocio de la aplicación, gestionando la actividad de los objetos de negocio para cumplir los objetivos de negocio iniciados a través de una petición de página desde el cliente Web. La clase *ClientPage* tiene un comportamiento y relaciones completamente diferentes. Se relaciona con el navegador Web a través de *Document Object Model* (DOM), el código JavaScript y en general con cualquier *plug-in* que la página especifique. Como se comentó, las clases *ServerPage* y *ClientPage* son dos abstracciones de una misma página Web, que en UML-WAE se deben relacionar a través de una asociación dirigida y estereotipada con *Build* indicando que la página de servidor construye la página cliente (aunque también puede haber páginas clientes que no han sido generadas a partir de una página de servidor).

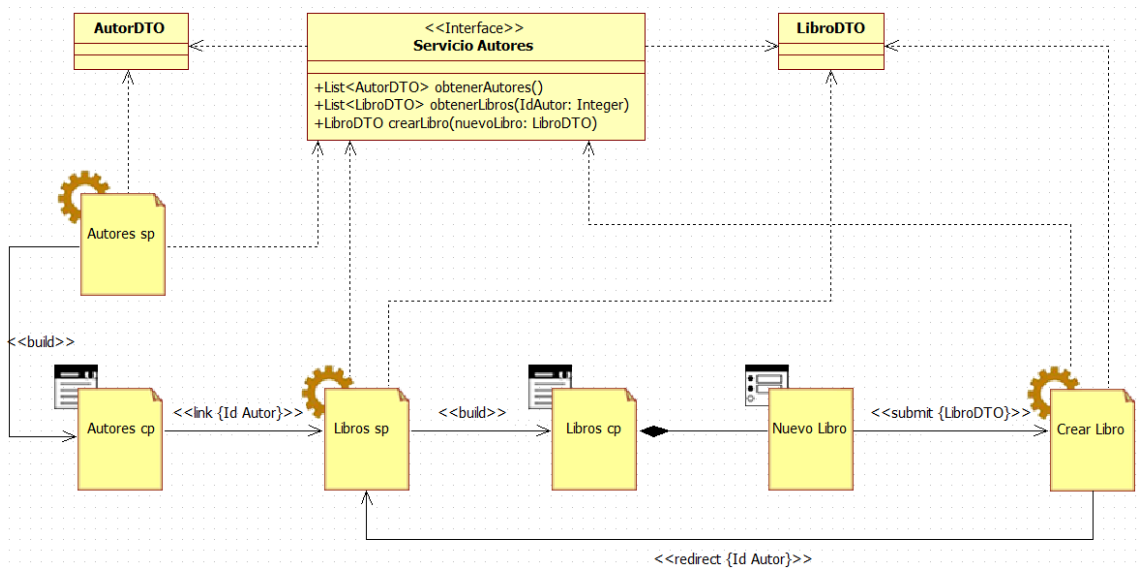
Otra relación común entre páginas Web es el enlace. Un enlace representa un camino navegacional a través de la aplicación Web. En UML-WAE esta relación se expresa a

través de una asociación dirigida y estereotipada como *Link*. Esta asociación siempre debe originarse en una clase *ClientPage* y finalizar en una clase *ClientPage* o en una clase *ServerPage*. Para expresar los parámetros que pueden ser pasados en un enlace se hace uso de valores etiquetados como una lista de pares clave-valor.

Los formularios son el principal mecanismo de entrada de datos en una aplicación Web. En UML-WAE los formularios se representan por medio de clases estereotipadas como *Form*, deben pertenecer a una clase *ClientPage* y deben especificar la clase *ServerPage* a la cual enviarán la información a través de la relación dirigida y estereotipada como *Submit*. UML-WAE también incluye elementos para caracterizar la interfaz de usuario. Así, clases estereotipadas con la palabra *Frameset* caracterizan ventanas y clases estereotipadas con la palabra *Target* caracterizan regiones de esas ventanas en las que se van a mostrar páginas destino de algún enlace.

La Figura 2.14 muestra un modelo UML-WAE con los principales estereotipos comentados. Como se puede ver en la Figura, el modelo UML-WAE representa un mapa navegacional en el que los componentes de la capa de presentación se relacionan con componentes de otras capas de la aplicación. Nótese como las páginas de servidor dependen de un elemento de negocio, el cual puede ser implementado como un servicio de aplicación (*application service*), un delegado de negocio (*business delegate*) que accede a un servicio remoto o cualquier otro patrón o combinación de patrones del catálogo de patrones arquitectónicos y de diseño que implementen la lógica requerida por la aplicación.

Figura 2.14: Ejemplo de un modelo UML-WAE.



2.8.2 El Perfil UML-WAE

Como ya se ha comentado, el perfil UML-WAE se expresa en términos de estereotipos, valores etiquetados y restricciones que extienden la notación de UML. Estos elementos permiten definir nuevos tipos de constructores que pueden ser usados para modelar aplicaciones Web. En la Figura 2.15 se muestra los estereotipos del perfil UML-WAE que han sido relevantes para este trabajo utilizando la notación gráfica propuesta en *UML Infraestructure*. Como puede observarse en la figura, cada estereotipo se define gráficamente dentro de un paquete estereotipado con *profile*, tienen un nombre y se asocian a un conjunto de elementos del meta-modelo de UML. Tal y como se indica, en el perfil UML-WAE las clases y las asociaciones de UML se adaptan al dominio Web.

Como se muestra en la Figura 2.15, para los fines de este trabajo se ha agregado a los estereotipos *Link* y *Submit* un valor etiquetado denominado *Target* de tipo String que indica la región destino de las páginas que reciben esta asociación. De igual forma al estereotipo *Target* se ha agregado el valor etiquetado *DefaultPage* también de tipo String que indica la página por defecto asignada a cada clase estereotipada con *Target*. Por la relevancia que tiene UML-WAE para este trabajo, las Tablas 2.1-2.10 describen en detalle los estereotipos, valores etiquetados y restricciones definidos en el perfil UML-WAE y que se muestran en la Figura 2.15.

Figura 2.15: Perfil UML-WAE en notación UML Infrastructure.

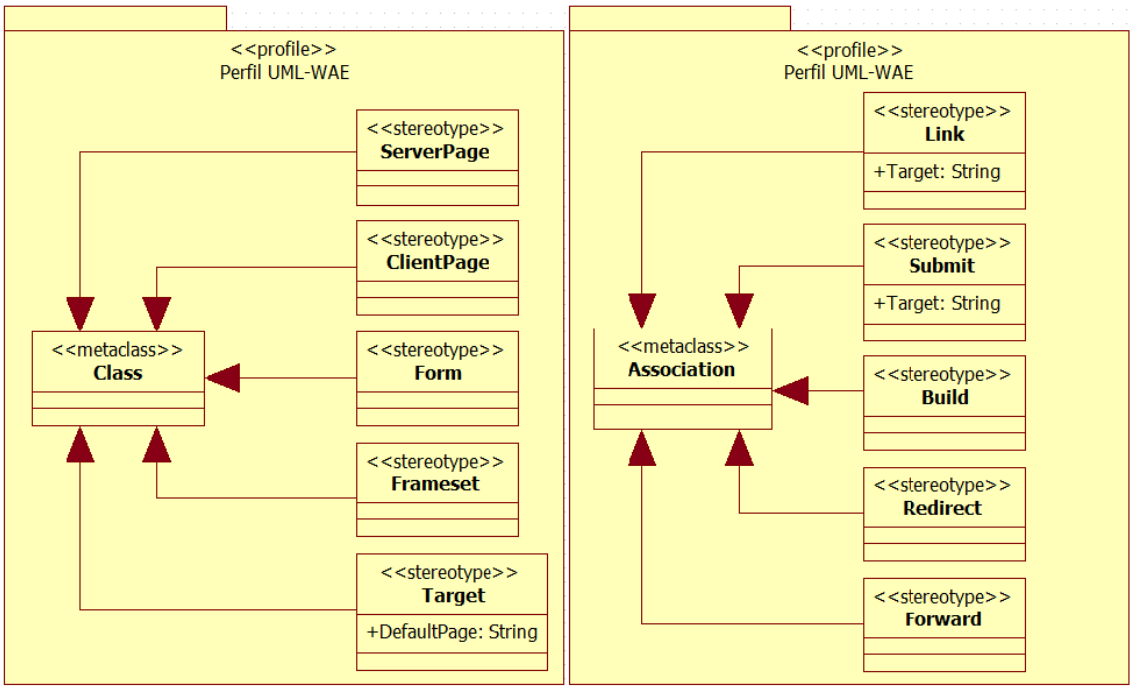


Tabla 2.1: Estereotipo WAE ServerPage

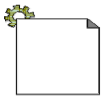
Nombre	**ServerPage**
Clase meta-modelo	Clase
Descripción	Una página de servidor representa una página Web que tiene scripts que son ejecutados por el servidor. Estos scripts operan con recursos en el servidor (bases de datos, lógica de negocio, sistemas externos, etc). Los métodos de la clase representan las funciones en el script, y sus atributos representan las variables que son visibles en el ambito de la página (accesible por todas las funciones en la página).
Icono	
Restricciones	Las páginas del servidor sólo pueden tener relaciones con objetos en el servidor.
Valores etiquetados	Artefacto de scripting o lenguaje o artefacto que debe ser usado para ejecutar o interpretar esta página (ASP, JSP, servlets, Perl, etc.)

Tabla 2.2: Estereotipo WAE ClientPage

Nombre	**ClientPage**
Clase meta-modelo	Clase
Descripción	Una página de cliente se corresponde con una página Web estructurada con un lenguaje de marcado como HTML o XML que contiene una mezcla de datos y presentación. Las páginas del cliente son presentadas por navegadores Web y pueden contener scripts que son interpretados y ejecutados por el navegador. Las páginas del cliente pueden asociarse con otras páginas del cliente o del servidor.
Icono	
Restricciones	Ninguna
Valores etiquetados	TitleTag – El título de la página desplegado por el navegador. BaseTag – La URL base para obtener la página. BodyTag – El conjunto de atributos para la etiqueta <body> que establece sus valores por defecto.

Tabla 2.3: Estereotipo WAE Form

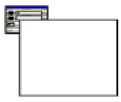
Nombre	**Form**
Clase meta-modelo	Clase
Descripción	Un formulario es una colección de campos de entrada que son parte de una página de cliente. Esta clase representa directamente a la etiqueta <form> de HTML. Sus atributos representan los campos de entrada del formulario (cajas de texto, áreas de texto, botones de selección, campos ocultos, etc.). Un formulario no puede tener operaciones, cualquier operación que interactúe con el formulario debe estar definida en la página cliente que contenga al formulario.
Icono	
Restricciones	Ninguna
Valores etiquetados	Método usado para enviar los datos a la URL, puede ser GET o POST

Tabla 2.4: Estereotipo WAE Frameset

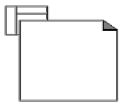
Nombre	**Frameset**
Clase meta-modelo	Clase
Descripción	Esta clase representa directamente a la etiqueta <frameset> de HTML. Esta página divide la interfaz de usuario en regiones rectangulares, en cada una de las cuales se puede presentar una página de cliente diferente.
Icono	
Restricciones	Debe contener al menos una página de cliente o región.
Valores etiquetados	Filas (Rows): número de filas en las cuales la interfaz es dividida. Columnas (Cols): número de columnas en las cuales la interfaz es dividida.

Tabla 2.5: Estereotipo WAE Target

Nombre	**Target**
Clase meta-modelo	Clase
Descripción	Una clase que representa una región con nombre dentro de un frameset

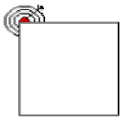
Icono	
Restricciones	El nombre de una clase estereotipada debe ser único para cada cliente del sistema.
Valores etiquetados	Fila (Row): el número de fila en el cual la región debe ser dibujada. Columna (Col): el número de columna en el cual la región debe ser dibujada.

Tabla 2.6: Estereotipo WAE Link

Nombre	**Link**
Clase meta-modelo	Asociación
Descripción	Representa una relación dirigida desde una página cliente a otra página cliente o página de servidor, se corresponde directamente con la etiqueta <a> de HTML.
Restricciones	Ninguna
Valores etiquetados	Parámetros: contiene los parámetros pasados con la petición http. Este valor etiquetado puede tener formato de cadena y estar codificado.

Tabla 2.7: Estereotipo WAE Build

Nombre	**Build**
Clase meta-modelo	Asociación
Descripción	Representa una relación dirigida desde una página de servidor hasta una página cliente. Esta relación identifica el resultado HTML o XML de la ejecución de una página de servidor. Una página de servidor puede construir muchas páginas cliente, pero una página cliente puede ser construida sólo por una página de servidor.
Restricciones	Ninguno.
Valores etiquetados	Ninguno.

Tabla 2.8: Estereotipo WAE Submit

Nombre	**Submit**
Clase meta-modelo	Asociación
Descripción	Representa una relación dirigida entre un formulario y una página de servidor. La página de servidor procesa los datos que el formulario le envía, que por lo general se corresponden con los campos de la entrada.
Restricciones	Ninguno.
Valores etiquetados	Ninguno.

Tabla 2.9: Estereotipo WAE Forward

Nombre	**Forward**
Clase meta-modelo	Asociación
Descripción	Representa una relación dirigida desde una página de servidor a otra página de servidor o página cliente representando la delegación del procesamiento de una petición de un cliente.
Restricciones	Ninguno.
Valores etiquetados	Ninguno.

Tabla 2.10: Estereotipo WAE Redirect

Nombre	**Redirect**
Clase meta-modelo	Asociación
Descripción	Representa una relación dirigida desde una página de cliente o una página de servidor a otra página cualquiera. Esta asociación indica un comando para que el cliente solicite otro recurso.
Restricciones	Ninguno.
Valores etiquetados	Ninguno.

2.8.3 UML-WAE y E-WAE

UML-WAE permite el diseño y desarrollo de aplicaciones Web reutilizando todos los beneficios de UML y siguiendo su misma filosofía: la semántica de sus estereotipos se define en términos de código y los modelos se basan en diagramas UML de clases con clases directamente traducibles a código orientado a objetos y diagramas UML de

secuencia con instancias de esas clases representado las interacciones entre objetos. Por tanto, los modelos UML-WAE son representaciones UML de código orientado a objetos que pueden ser desarrollados, mantenidos e integrados con otros modelos UML que representen otros componentes de una aplicación en cualquier herramienta UML CASE genérica.

De esta forma, la integración de los modelos UML-WAE de la capa de presentación Web con los modelos UML que representan las otras capas de una aplicación puede implementarse usando toda la potencia de los patrones arquitectónicos y de diseño definidos en los catálogos de patrones multicapa, los cuales se describen normalmente en UML. De esta forma, se puede afirmar que UML-WAE promueve el desarrollo de aplicaciones Web siguiendo la arquitectura multicapa y que no requiere de herramientas CASE propietarias para desarrollar sus diseños.

El hecho de que los modelos UML-WAE sean representaciones directas de código permite la especificación detallada del diseño arquitectónico, lo cual hace que UML-WAE sea adecuado para modelar aplicaciones Web complejas, las cuales requieren la implementación de arquitecturas con múltiples capas. Sin embargo, esta característica de UML-WAE puede hacer que sus modelos sean difíciles de entender, ocultando la visión general de una aplicación y por tanto, haciendo que UML-WAE sea adecuada sólo para diseñadores y desarrolladores de aplicaciones Web.

Por tanto, desde el punto de vista de este trabajo, UML-WAE tiene dos limitaciones importantes: (i) sus diseños son dependientes de una arquitectura e incluyen conceptos de programación, por ejemplo, la presencia de controladores o el enlace directo entre páginas; y (ii) la notación no soporta los componentes presentes en frameworks empresariales para el desarrollo de la capa de presentación Web, como JSF o ASP.NET MVC. De esta forma, UML-WAE no proporciona las ventajas de las notaciones abstractas de alto nivel y al mismo tiempo, sus modelos ya no son adecuados para caracterizar los componentes de la capa de presentación Web de aplicaciones modernas, las cuales normalmente se desarrollan usando frameworks empresariales como los anteriormente mencionados. Desde este punto de vista, podemos afirmar que con respecto a las aplicaciones modernas, los modelos UML-WAE están lejos del nivel de implementación y sus beneficios, por tanto, se limitan a la producción de diseños de representación sin asistir el proceso de desarrollo.

E-WAE sigue la filosofía de UML-WAE definiendo un número mínimo de estereotipos para caracterizar la capa de presentación Web y promoviendo el uso de UML para modelar el resto de capas de una aplicación, heredando todos los beneficios de UML-WAE comentados anteriormente. Sin embargo E-WAE subsana las dos limitaciones de UML-WAE que hemos mencionado en el desarrollo de aplicaciones Web modernas. Primero, E-WAE añade los beneficios de las notaciones de alto nivel agregando un nivel de abstracción a UML-WAE. E-WAE reutiliza los estereotipos que pueden ser validados a nivel de usuario y define nuevos estereotipos que representan conceptos de aplicaciones Web modernas independientemente de arquitecturas concretas y plataformas de desarrollo. Y segundo, proporciona dos extensiones de UML-WAE que caracterizan los componentes implementados en los frameworks JSF y ASP.NET MVC que han sido llamados WAE4JSF y WAE4.NET respectivamente. Estas dos extensiones permiten el diseño de la capa de presentación Web usando los componentes implementados en los frameworks facilitando la generación del código a partir de ellos de una forma transparente, y de esta forma asistiendo el desarrollo de las aplicaciones.

Además E-WAE proporciona un conjunto de reglas explícitas que permiten la transformación de los modelos E-WAE a modelos WAE4JSF o WAE4.NET y de estos a código. Estas reglas permiten la transformación manual de los modelos independizando el proceso de desarrollo de herramientas propietarias y pueden ser usadas como base para la implementación de transformaciones automáticas dentro de un proceso MDD. En este sentido, se podría afirmar que E-WAE adapta la filosofía de UML-WAE al desarrollo de aplicaciones empresariales modernas.

2.9 Análisis de las diferentes aproximaciones

En esta sección se comparan las diferentes aproximaciones analizadas, así como E-WAE. Para facilitar este análisis se han categorizado estas aproximaciones en dos grupos principales: orientadas al modelado y plataformas de desarrollo visual.

Las aproximaciones orientadas al modelado proporcionan notaciones bien definidas para el modelado de aplicaciones Web. Entre ellas están las metodologías de la Ingeniería Web tales como WebML, OOH4RIA, y UWE y sus extensiones (que en este trabajo son consideradas como parte de UWE) UWE4JSF (Kroiss et al., 2009) y UWESec (Busch et al., 2011; Busch et al., 2012). Como se comentó en la sección anterior, MockupDD es una aproximación que extiende a las metodologías de Ingeniería Web, incorporando principios de desarrollo ágil a ellas, por tanto en el siguiente análisis, MockupDD forma

parte de ellas. IFML y NMMp pueden ser consideradas como propuestas recientes de este tipo de metodologías.

Las plataformas de desarrollo visual pueden ser vistas como entornos de programación visual para lenguajes específicos que configuran el código usando diagramas. Por tanto, estas plataformas tienen como objetivo abstraer a los desarrolladores del código y otros detalles técnicos como la inclusión explicita de patrones arquitectónicos y de diseño, para acelerar los tiempos de desarrollo. Aunque estas plataformas no están centradas en el modelado, ni en la generación de mockups, se analizan en este trabajo porque proporcionan una pseudo-notación gráfica para desarrollar aplicaciones Web. Entre las plataformas más relevantes de este tipo están Mendix y Outsystems.

La Tabla 2.11 muestra un resumen de las principales características de estas aproximaciones. WebML, UWE y OOH4RIA, y las plataformas de desarrollo visual, tienen la capacidad de generar aplicaciones Web completas por medio de los poderosos elementos de modelado que definen y sus herramientas generadoras de código. IFML se centra en la especificación de interfaces de usuario de cualquier tipo de dispositivo/plataforma. NMMp y UML-WAE se centran en el modelado y desarrollo de la capa de presentación de las aplicaciones Web, y E-WAE, por el soporte que da al desarrollo con frameworks industriales, se centra en el desarrollo de la capa de presentación Web de aplicaciones empresariales modernas.

Las metodologías de Ingeniería Web como WebML, UWE y OOH4RIA, y las plataformas de desarrollo visual, se centran en el desarrollo de modelos/diagramas de alto nivel de abstracción, los cuales son utilizados como modelos fuente a partir de los cuales se genera el código de las aplicaciones. Esta generación se ejecuta por medio de herramientas que actúan como generadores de código propietario (*black-box wizards*) que no permiten controlar la forma en el que se genera éste y vinculan el mantenimiento de las aplicaciones a ellos.

Por otro lado, los diagramas proporcionados por las plataformas de desarrollo visual no pueden considerarse modelos de software, porque carecen de una definición formal a través de meta-modelos o notaciones bien definidas. Los modelos NMMp y UML-WAE, son diseños abstractos que están aún lejos del nivel de implementación, ya que no caracterizan componentes de plataformas específicas. Por el contrario, aunque los modelos E-WAE son modelos PIM, E-WAE proporciona una semántica orientada a código de sus elementos a través de la definición de los modelos PSM WAE4x y las reglas

de transformación explícitas PIM2PSM y PSM2Code que evitan acoplar la generación y mantenimiento de código a herramientas específicas. OOH4RIA también define modelos PSM para representar el código que se ejecuta del lado del cliente, pero dejando que el código del lado del servidor sea generado directamente a partir de los modelos PIM.

La mayoría de las aproximaciones analizadas no promueven el desarrollo siguiendo la filosofía UML. OOH4RIA y las plataformas de desarrollo visual utilizan notaciones propietarias que no están basadas en UML. Aunque IFML, WebML y UWE proporcionan perfiles UML, estos son tan complejos que en la práctica son nuevas notaciones. La complejidad de estos perfiles se refleja en el alto número de estereotipos que definen, más de 50 por notación, y su naturaleza abstracta, los cuales no representan clases directamente traducibles a código orientado a objetos, sino elementos del dominio Web. De hecho, los modelos que proporcionan estas notaciones para definir la estructura estática y dinámica no están basados en diagramas UML de clases y UML de secuencia respectivamente, como determina el Proceso Unificado de Desarrollo (Jacobson et al., 1999). Por el contrario, UML-WAE, NMMp y E-WAE siguen la filosofía de desarrollo de UML, el número de estereotipos que definen es mínimo para caracterizar los componentes de la capa de presentación Web, la semántica de estos estereotipos está cercana a conceptos de programación (y no a conceptos del dominio Web) y los diagramas que proponen están basados en los diagramas UML de clases y UML de secuencia. Estos enfoques promueven el modelado del resto de capas con UML estándar y permiten la integración de la capa de presentación con el resto de capas modelando una aplicación empresarial completa.

UML-WAE, NMMp y E-WAE son los únicos enfoques que han sido desarrollados con el concepto de la arquitectura multicapa en mente, centrándose solo en la capa de presentación Web. Esta capa puede ser diseñada y generada de forma desacoplada del resto de capas, a través del uso de *Objetos de Transferencia de Datos* (DTO), el mecanismo básico para intercambiar información entre capas en la arquitectura multicapa. No obstante, con estos enfoques, la capa de presentación Web puede integrarse con el resto de capas de una aplicación siguiendo los patrones multicapa. WebML, UWE, OOH4RIA y hasta cierto punto IFML, definen elementos de modelado inspirados en las primitivas de acceso RMM (Isakowitz et al., 1995): son primitivas que, incluidas en modelos navegacionales dan acceso directo a entidades del modelo de datos. Por tanto, en estos enfoques, los modelos de capa de presentación se acoplan directamente a la capa de recursos, obviando la capa de negocio, la cual es clave en el desarrollo de aplicaciones

empresariales. Así, estos enfoques pueden encajar mejor en el desarrollo de aplicaciones dirigidas por datos que en el desarrollo de aplicaciones empresariales. Las plataformas de desarrollo visual, a través de sus diagramas, intentan abstraer lo más posible los detalles técnicos como la generación de código y la implementación de patrones arquitectónicos y de diseño. Por tanto, no tienen en cuenta la mantenibilidad de las aplicaciones.

El diseño arquitectónico garantiza la mantenibilidad de las aplicaciones por medio de la inclusión explicita de patrones de diseño en los modelos, incluyendo aquellos relacionados con seguridad, administración de transacciones e implementación de bloqueos (Fowler, 2002; Alur et al., 2003). La inclusión de estos patrones en los modelos es una decisión de diseño clave en el desarrollo de aplicaciones empresariales, debido a que facilitan su entendimiento, proporcionando una visión gráfica del código, lo cual, favorece su mantenibilidad, uno de los objetivos del diseño orientado a objetos y uno de los atributos fundamentales del software de acuerdo a ISO/IEC 9126 (Jung et al., 2004).

Nótese que esta característica se refiere a la presencia explicita de los patrones en los modelos y no solo a la posibilidad de soportar las características en las aplicaciones finales. Por ejemplo, como se comentó en la Sección 2.4, IFML soporta la definición de transacciones por medio de elementos de modelado que agrupan a otros elementos que involucran acceso a bases de datos (ver Figura 2.8). Sin embargo, dado que estos son elementos de alto nivel de abstracción, solo pueden modelar las transacciones a nivel descriptivo, reflejando la reacción del sistema ante posibles fallos en el acceso a datos. Por tanto, la implementación de esta característica depende íntegramente de la herramienta generadora de código, la cual, por su puesto, puede implementar ciertos patrones predefinidos relacionados con el manejo de transacciones. Sin embargo, estos patrones no se reflejan en los modelos y generalmente, suelen cubrir los casos más comunes no pudiendo ser adaptados para solucionar un problema de dominio específico. Esta es una característica presente en todas las metodologías que generan modelos de alto nivel de abstracción sin una semántica orientada al código.

OOH4RIA considera la inclusión de patrones arquitectónicos en sus diagramas de componentes, pero no considera la inclusión de patrones de diseño en los diagramas de más bajo nivel donde deben ser incluidos. UML-WAE, NMMp y E-WAE, debido a que siguen la filosofía de desarrollo UML, permiten incluir en sus modelos todo el catálogo de patrones arquitectónicos y de diseño en sus diagramas UML de clase y UML de secuencia.

Finalmente, la generación de código puede ser independiente de herramientas específicas si se especifica conforme a los estándares OMG, pudiendo ser reproducida en cualquier herramienta compatible con estos estándares. Ninguna de las aproximaciones analizadas cumple con esta característica. IFML, las metodologías de la Ingeniería Web y las plataformas de desarrollo visual utilizan generadores de código propietarios (*black-box wizards*). NMMp y UML-WAE no tienen capacidad de generación de código. En E-WAE, además de manera automática, la generación de código puede ser ejecutada manualmente debido a la definición de una semántica orientada a código de los modelos PSM y el conjunto de reglas de transformación bien definidas. Aunque estas reglas pueden ser implementas de acuerdo a la especificación MOF de transformaciones modelo a texto, esta implementación aún no ha sido desarrollada, utilizándose en la actualidad reglas de transformación propietarias de la herramienta Borland Together, y una herramienta propietaria para el despliegue del código en sus respectivos IDEs.

Tabla 2.11: Resumen de las principales aproximaciones para el desarrollo Web

Notación	Orientad. al modelado /desarro.	Centrada en	Modelos PIM/PSM	Transformac. PIM2PSM	Perfil UML				Principal uso: perfil UML o notación propietar.	Generac. de código automát.	Modelado RMM o modelado multicapa	Inclusión explicita de patrones arquitectón. y de diseño	Modelado de elementos RBAC	Generac. portable de código
					Número de estereotipos	¿Clases UML representa código OO?	Tipo principal de diagrama estructural	Tipo principal de diagrama dinámico						
IFML	Modelado	Interfaces de usuario	PIM	No	75	Parcial	Componente	Componente	Propietaria	Parcial	RMM	Fuera de la capa de presentación	No	No
WebML	Modelado	Aplicaciones Web	PIM	No	Más de 50	No	Componente	Componente	Propietaria	Sí	RMM	No	No	No
UWE	Modelado	Aplicaciones Web	PIM	No	Más de 50	No	Clases	Actividad	UML	Sí	RMM	No	Sí	No
OOH4RIA	Modelado	Aplicaciones Web	PIM	Parcial	n/a	n/a	Componente	n/a	Propietaria	Sí	Mixta	Parcial	No	No
WAE	Modelado	Capa de presentación Web	PIM	n/a	12	Parcial	Clases	Secuencia	UML	No	Multicapa	Sí	No	No
NMMp	Modelado	Capa de presentación Web	PIM	No	9	n/a	n/a	n/a	UML	No	Multicapa	Fuera de la capa de presentación	No	No
E-WAE/ WAE4x	Modelado	Capa de presentación Web (empresarial)	PIM y PSM	Sí	22	Parcial	Clases	Secuencia	UML	Sí	Multicapa	Sí	Sí	No
Plataformas visuales(Mendix-Outsystems)	Desarrollo	Aplicaciones Web	n/a	n/a	n/a	n/a	n/a	n/a	Propietaria	Sí	Propietaria	No	No a nivel modelo	No

3 NMM PROFILE (NMMP)

3.1 Introducción

Esta sección describe la notación NMMp, una notación previa a la notación E-WAE. Se incluye en esta memoria porque fue un paso importante que ayudó a concebir la notación E-WAE. Así, sin la experiencia proporcionada durante el desarrollo y aplicación de la notación NMMp, no hubiera podido ser posible desarrollar la notación E-WAE. Aunque las primitivas de modelado NMMp y E-WAE difieren notablemente, ambas comparten la misma filosofía de uso: proporcionar modelos PIM para caracterizar la capa de presentación de aplicaciones Web, para posteriormente generar modelos PSM en términos de diagramas WAE o WAE4x.

NMMp (Cortes, 2011; Cortés & Navarro 2014) toma como base la notación de diseño NMM (Navarro et al., 2008) desarrollada en la academia para caracterizar de forma abstracta mapas navegacionales con independencia de detalles arquitectónicos y conceptos de programación. NMM se centra en definir formalmente semánticas de navegación (Stotts et al., 1998) para aplicaciones Web, por tanto, NMM permite especificar y comprobar mediante sus modelos las propiedades de navegación de una aplicación. Sin embargo, Aunque NMM tiene un desarrollo formal, su notación visual no está basada en UML y por tanto, requiere de su propia herramienta CASE, lo cual es un serio obstáculo para su aplicación en el desarrollo industrial. NMMp ha sido diseñada priorizando la usabilidad sobre la definición formal. Así, NMMp puede ser vista como la evolución de NMM a versión UML que permite explotar todas sus características en herramientas UML CASE genéricas, incluyendo la capacidad de transformar automáticamente los modelos NMMp en modelos UML-WAE a través de la ejecución de

un conjunto de reglas de transformación basadas en el estándar QVT (*Query/View/Transformation*) (OMG, 2008). Además, con NMMp los modelos UML-WAE obtenidos pueden ser directamente integrados con el diseño UML del resto de componentes de una aplicación Web dentro de la herramienta UML CASE seleccionada.

Aunque la potencia de modelado de NMM y NMMp es prácticamente equivalente, existen cambios considerables en los modelos formales subyacentes. Por ejemplo, NMM define una función computacional para una aplicación *A* como $comp_A$*: Anchor*$_C$ *x Input* → *Page x AI*. Lo cual básicamente establece que cada *ancla* capaz de iniciar un proceso computacional en una aplicación *A*, con un conjunto de valores de entrada, genera una página destino con seis conjuntos diferentes de anclas (representados por el conjunto *AI*) (Navarro et al., 2008). Este tipo de especificaciones que son necesarias para proporcionar semánticas formales de navegación imponen importantes restricciones en los diagramas visuales. Por otro lado, NMMp no considera este tipo de formalizaciones permitiendo la libre definición de anclas, y por tanto, el diseño intuitivo de diagramas visuales, aunque estos no puedan ser utilizados en procesos de comprobación formal. Sin embargo, NMMp conserva las definiciones formales de NMM que permiten la caracterización de sus elementos con UML y la definición las transformaciones automáticas a elementos definidos en UML-WAE.

Por tanto, con respecto a NMM, NMMp: (i) introduce modificaciones en el modelo de definición formal de NMM para hacerlo más usable; (ii) evoluciona NMM desde una notación formal a una extensión UML incrementando su aplicabilidad; (iii) modifica la apariencia visual de la notación; (iv) implementa transformaciones automáticas entre modelos NMMp y modelos UML-WAE usando herramientas comerciales y por tanto, proporcionando un entorno de desarrollo profesional para la industria; y (v) combina las ventajas de las notaciones de alto nivel promovidas por las metodologías de ingeniería Web con la implementación de patrones arquitectónicos y de diseño que han sido probados en la industria durante años (Alur et al., 2003; Fowler, 2002).

NMMp es una notación desarrollada con el concepto de la arquitectura multicapa (Alur et al., 2003) en mente. Dentro de esta arquitectura, NMMp se ocupa de modelar la capa de presentación de las aplicaciones Web. En NMMp dicha capa está compuesta por la estructura navegacional de la aplicación, la descripción de la interfaz de usuario en términos de regiones y la relación entre ambas. La estructura navegacional se representa a través de *diagramas de páginas* que describen todas las secuencias posibles de páginas Web que pueden ser visitadas por los usuarios. La interfaz de usuario se modela a través

de *diagramas de regiones* que muestran como se dividen las ventanas de la interfaz. La relación entre la estructura navegacional y la interfaz de usuario se representa a través de *diagramas de mezcla*, los cuales describen el acceso navegacional de los usuarios a las páginas a través de la interfaz de usuario.

NMMp permite obtener al mismo tiempo los beneficios de las notaciones de diseño de alto y bajo nivel. NMMp como notación de alto nivel caracteriza los elementos principales de las aplicaciones Web omitiendo detalles arquitectónicos (por ejemplo, la presencia o no de un controlador), haciendo que los modelos sean sencillos y muestren una visión general de la aplicación. NMMp como notación de bajo nivel, dado que está basada en una notación con semántica formal, puede ser fácilmente transformada a modelos UML-WAE (Conallen, 1999) que caracterizan diseños arquitectónicos detallados. Así, dependiendo de la complejidad de una aplicación Web, se puede elegir la arquitectura más adecuada en el momento de ejecutar las transformaciones de los modelos NMMp a modelos UML-WAE. De esta forma NMMp combina las ventajas de las notaciones de alto nivel promovidas por las metodologías de Ingeniería Web con la implementación de patrones arquitectónicos y de diseño promovidos en la industria.

Como lo comentamos anteriormente, NMMp se centra en el modelado de la capa de presentación de las aplicaciones Web, no se ocupa de modelar los componentes del lado del servidor responsables del enrutamiento de páginas, componentes que ejecutan procesos computacionales y los datos que fluyen entre las páginas y estos procesos. Estos componentes se generan en el proceso de transformación automático a modelos UML-WAE de acuerdo a la arquitectura seleccionada. A continuación se detalla el proceso de desarrollo del perfil NMMp.

3.2 Desarrollo del perfil NMMp

El principal objetivo de NMMp es modelar la capa de presentación Web con el concepto de usabilidad en mente. En el contexto de este trabajo, usabilidad debe interpretarse como la capacidad de la notación para soportar el desarrollo de aplicaciones empresariales. Como se ha comentado, la adecuada definición de la arquitectura de software y su modelado, son aspectos clave en el desarrollo de este tipo de aplicaciones. Respecto a la arquitectura, nótese que NMMp se ajusta naturalmente al desarrollo siguiendo la arquitectura multicapa. Respecto al modelado, basar la notación en UML, el estándar *de facto* para modelar aplicaciones empresariales, fue la opción más directa, no sin antes explorar las diferentes alternativas en este ámbito.

UML estándar se basa principalmente en conceptos de diseño orientado a objetos. Sin embargo, la capa de presentación Web incluye elementos, como páginas y enlaces, entre otros, que no pueden ser caracterizados con este tipo de conceptos. Por ejemplo, UML estándar no permite modelar qué componentes de una página Web se ejecutan del lado del servidor y qué componentes se ejecutan del lado del cliente. Así mismo, UML estándar no permite modelar los enlaces navegacionales entre las páginas Web junto con la información transportada en ellos.

Para modelar este tipo de conceptos específicos de un dominio (en este caso, del dominio Web), la ingeniería dirigida por modelos (*Model-Driven Engineering*, MDE) promueve principalmente dos técnicas: el uso de lenguajes de modelado de dominio específicos (*Domain Specific Modeling Languages*, DSML) basados en MOF, y el uso de perfiles UML. La primera técnica, como su nombre lo indica, consiste en definir lenguajes específicos de un dominio, cuyos conceptos son formalmente definidos a través de un meta-modelo MOF. Los lenguajes definidos con esta técnica tienen su propia representación, gráfica o textual, y requieren contar con su propio conjunto de herramientas para soportarla. Algunos ejemplos de lenguajes específicos del dominio Web son OOH4RIA y WebML.

La segunda técnica, el uso de perfiles UML, consiste en extender UML con los conceptos del dominio específico. Un modelo extendido con un perfil UML permite caracterizar los conceptos del dominio utilizando los propios mecanismos de extensión de UML. Entre los perfiles UML más reconocidos en el dominio Web está UML-WAE. Como se describió en el Capítulo 2, la mayoría de las metodologías de la Ingeniería Web, usan las dos técnicas para modelar los conceptos del dominio Web, sin embargo, en nuestra opinión, a un nivel de abstracción que no es el adecuado para el desarrollo de aplicaciones empresariales. Desde el punto de vista del presente trabajo, la selección de la técnica adecuada para este tipo de aplicaciones, depende de tres aspectos clave: la disponibilidad de herramientas de modelado, la aceptación técnica por parte de los desarrolladores y la capacidad para integrase con otros modelos estándar.

En este sentido, NMMp como perfil UML, puede aprovechar todas las ventajas del estándar, incluyendo el soporte de todas las herramientas UML CASE genéricas y todo el conocimiento y experiencia acumulados por años en los sectores académico e industrial. Por tanto, un modelo UML extendido con el perfil NMMp puede llegar a ser intuitivo para los desarrolladores y puede ser generado y mantenido en herramientas ya existentes. Así mismo, NMMp como perfil UML, permite modelar los aspectos

específicos del dominio Web, sin obstaculizar la necesidades de integración con otros sistemas, que generalmente demandan las aplicaciones empresariales.

La ventaja más importante de NMMp como lenguaje de modelado específico podría ser la facilidad de desarrollar generadores automáticos de código a partir de sus modelos (Weisemöller & Schürr, 2007). Sin embargo, esta característica convertiría a NMMp en otra notación propietaria que requiere de un generador de código necesario para el mantenimiento de las aplicaciones. La generación automática de código es una característica muy valiosa en el proceso de desarrollo de software, pero, en nuestra opinión, no a expensas de vincular el mantenimiento del código generado a una herramienta específica. En cualquier caso, el enfoque NMMp considera los diagramas de diseño como representaciones abstractas de código y no como simples representaciones visuales, por tanto, a través de las transformaciones a UML-WAE, se puede afirmar que el código es generado, aunque no la aplicación como tal. Además dado que UML-WAE es otro perfil UML, compartiría el mismo núcleo semántico que NMMp como perfil UML, facilitando la implementación de las reglas de transformación (OMG, 2013).

Después de analizar las dos técnicas para el modelado de los conceptos del dominio Web en el contexto del desarrollo de aplicaciones empresariales, se notó claramente que la definición de NMMp como perfil UML es la más adecuada. Navarro & Cortés (2011) describen la experiencia práctica durante el desarrollo del meta-modelo MOF y del perfil UML para NMMp. En las secciones siguientes se presenta el meta-modelo MOF que proporcionan una especificación precisa de los elementos definidos en NMMp. A continuación, basado en este meta-modelo, se presenta la definición del perfil UML para NMMp, con el objetivo de utilizar la notación en el desarrollo de aplicaciones empresariales.

3.2.1 Meta-modelo MOF para NMMp

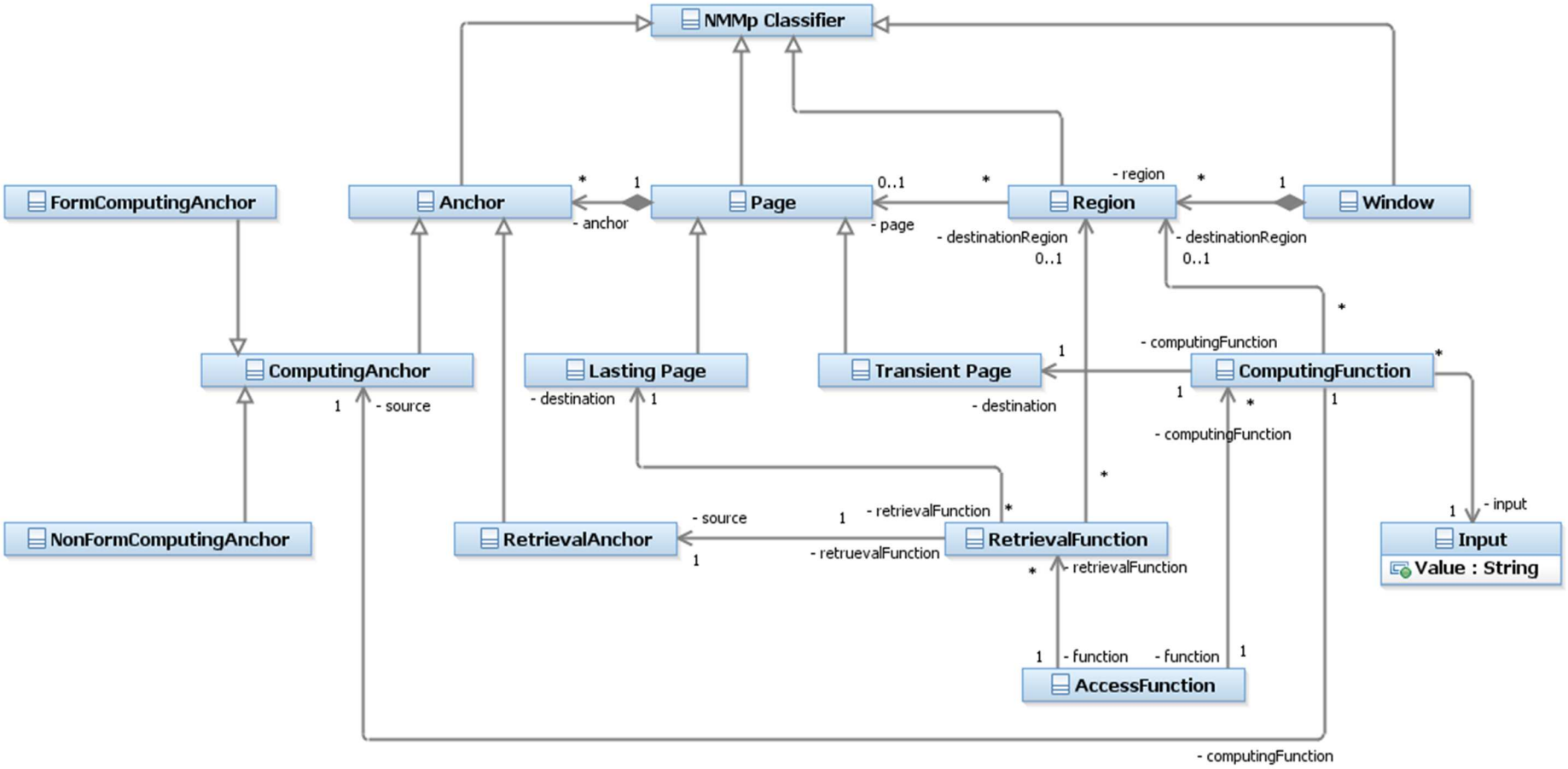
El meta-modelo MOF mostrado en la Figura 3.1 (Cortés & Navarro, 2014) proporciona una especificación precisa para los elementos definidos en el perfil NMMp. El elemento *Page* caracteriza las páginas que son administradas por un navegador Web, las cuales pueden ser de dos tipos: *Lasting Page* y *Transient Page*. Los primeros representan páginas que están completamente definidas antes de cualquier interacción con la aplicación, es decir no requieren de procesos computacionales para su elaboración. Los segundos representan páginas que son construidas a través de la ejecución de procesos

computacionales invocados por el servidor Web, es decir, son generadas dinámicamene como resultado de la interacción con la aplicación Web.

La estructura navegacional se modela usando los elementos *Anchor*, las cuales están contenidos en páginas Web fuente y dan acceso a páginas Web destino. Es decir, un *Anchor* representa un dispositivo dentro de una página Web que tiene la capacidad de iniciar una petición HTTP a un servidor Web, generando una respuesta del lado del servidor que produce la página Web destino. Como puede observarse en la figura, NMMp diferencia dos tipos de *Anchor*: *Retrieval Anchor* y *Computing Anchor*. Las primeras dan acceso, por medio de los elementos *Retrieval Function*, a las páginas *Lasting Page*. Las segundas, dan acceso, por medio de los elementos *Computing Function*, a las páginas *Transient Page*. Además los elementos *Computing Anchor* son clasificados como *Form Computing Anchor*, las cuales representan botones de envío dentro de formularios Web, y como *Non Form Computing Anchor*, las cuales representan dispositivos que generan peticiones desde cualquier proceso computacional diferente a los botones de envío.

Como se comentó anteriormente, los diagramas de páginas NMMp muestran las relaciones entre páginas en términos de transiciones de una página a otra ocultando los componentes del lado del servidor responsables de ejecutar el proceso navegacional. En NMMp, los elementos *Computing Anchor* representan estos componentes del lado del servidor. NMMp tampoco se ocupa del modelo de datos de las aplicaciones. Se asume que las páginas creadas dinámicamente incluyen la información extraída desde el modelo de datos. Más adelante, cuando los modelos NMMp se transforman a modelos UML-WAE que caracterizan arquitecturas concretas, los elementos *Computing Anchor* son la base para la generación de los componentes responsables de ejecutar la navegación y los objetos de transferencia de datos que relacionan el modelo de datos y la capa de presentación.

Figura 3.1: Meta-modelo MOF NMMp



Por otro lado, aunque NMMp no se ocupa del modelado de la estructura interna de las páginas, a través de los diagramas de regiones permite modelar las regiones que forman parte de las ventanas de la interfaz de usuario. Así, los diagramas de regiones NMMp pueden incluir la definición de regiones, ventanas y la relación de agregación entre ellas, que define cuando una región pertenece a una ventana. Para este propósito, NMMp define los elementos *Window* y *Region*, los cuales representan conceptos generales propios de un modelo independiente de la plataforma.

Así mismo, por medio de los diagramas de mezcla, NMMp permite modelar el acceso navegacional a las páginas de la aplicación a través de la interfaz de usuario. Como se puede ver en la figura, los elementos *Region* pueden tener asignadas páginas por defecto, lo cual implica que una región puede cargar una página predefinida cuando se invoca. Además, los elementos *Retrieval Function* y *Computing Function* que dan acceso a páginas destino en el proceso navegacional, tienen asignadas regiones por defecto, lo cual implica que una página destino puede ser cargada en una región determinada.

3.2.2 Perfil UML para NMMp

El meta-modelo MOF presentado en la sección anterior proporciona una descripción compacta de la sintaxis de la notación NMMp. Sin embargo, como se comentó, NMMp ha sido desarrollada con el concepto de usabilidad en mente. Por tanto, es necesario definir un perfil UML para habilitar el uso de NMMp en herramientas UML CASE genéricas, proporcionando un entorno de desarrollo profesional para el desarrollo industrial.

Fuentes & Vallecillo (2004) exponen algunas guías y recomendaciones prácticas para el desarrollo de perfiles UML. Cabe destacar que estas guías se han tomado como base para el desarrollo de todos los perfiles UML implementados en este trabajo. A continuación se comentan para el caso del perfil UML para NMMp:

1. Definir el meta-modelo que representa la capa de presentación Web utilizando los mecanismos del propio UML o MOF (clases, relaciones de herencia, asociaciones, etc.), de la forma usual como se haría si el objetivo fuese definir un lenguaje de modelado específico de dominio. Se debe incluir la definición de los elementos, las relaciones entre ellos, así como las restricciones que limitan el uso de estas entidades y de sus relaciones, tal y como se muestra en el meta-modelo MOF de la Figura 3.1.

2. Definir el perfil UML para NMMp con base en el meta-modelo definido en el punto anterior. Dentro del paquete *UML Profile* se incluye un estereotipo por cada uno de los elementos del meta-modelo que formarán parte del perfil. Estos estereotipos deben tener el mismo nombre que los elementos del meta-modelo, estableciéndose de esta forma una relación entre el meta-modelo y el perfil UML.
3. Es importante tener claro cuáles son los elementos del meta-modelo de UML que se requiere extender. Ejemplo de tales elementos son clases, asociaciones, atributos, operaciones, transiciones, paquetes, etc. El perfil NMMp extiende las clases y asociaciones UML.
4. Definir como propiedades de los estereotipos del perfil NMMp, los atributos que aparezcan en el meta-modelo. Incluir la definición de sus tipos, y sus posibles valores iniciales.
5. Definir las restricciones que forman parte del perfil NMMp a partir de las restricciones del dominio. Por ejemplo, las multiplicidades de las asociaciones que aparecen en el meta-modelo del dominio o las propias reglas de negocio deben traducirse en la definición de las correspondientes restricciones.

Las Tablas 3.1-3.9 describen cada uno de los estereotipos, valores etiquetados, restricciones e iconos que definen el perfil NMMp y que se corresponde con los conceptos descritos en el meta-modelo desarrollado en la sección anterior.

Tabla 3.1: Estereotipo NMMp Lasting Page

Nombre	**Lasting Page**
Clase metamodelo	Clase
Descripción	Páginas estáticas son aquellas páginas que están completamente definidas antes de cualquier interacción con la aplicación, es decir no requieren de procesos computacionales para su elaboración.
Icono	
Restricciones	Las páginas estáticas sólo pueden contener código en algún lenguaje de marcado como HTML ó XML y código Java Script.
Valores etiquetados	-

Tabla 3.2: Estereotipo NMMp Transient Page

Nombre	**Transient Page**
Clase metamodelo	Clase
Descripción	Páginas dinámicas son aquellas que son construidas a través de componentes computacionales invocados por el servidor Web. Es decir, son generadas por medio de procesos computacionales lanzados como resultado de la interacción con la aplicación Web.
Icono	
Restricciones	-
Valores etiquetados	-

Tabla 3.3: Estereotipo NMMp Retrieval Anchor

Nombre	**Retrieval Anchor**
Clase metamodelo	Clase
Descripción	Las anclas de recuperación son dispositivos dentro de una página Web que tiene la capacidad de iniciar una petición que retorna páginas estáticas.
Icono	
Restricciones	-
Valores etiquetados	Target contiene la región destino por defecto asignada a cada ancla.

Tabla 3.4: Estereotipo NMMp Form Computing Anchor

Nombre	**Form Computing Anchor**
Clase metamodelo	Clase
Descripción	Las anclas computacionales en formulario representan botones de envío dentro de *Web forms* que tiene la capacidad de iniciar una petición que retorna páginas dinámicamente creada.

Icono	
Restricciones	-
Valores etiquetados	Target contiene la región destino por defecto asignada a cada ancla.

Tabla 3.5: Estereotipo NMMp Non Form Computing Anchor

Nombre	**Non Form Computing Anchor**
Clase metamodelo	Clase
Descripción	Las anclas computacionales fuera de formulario representan dispositivos que generan peticiones desde cualquier proceso computacional diferente a los botones de envío
Icono	
Restricciones	-
Valores etiquetados	Target contiene la región destino por defecto asignada a cada ancla.

Tabla 3.6: Estereotipo NMMp Window

Nombre	**Window**
Clase metamodelo	Clase
Descripción	Representa cada una de las ventanas en la interfaz gráfica de usuario de la aplicación Web.
Icono	
Restricciones	-
Valores etiquetados	-

Tabla 3.7: Estereotipo NMMp Region

Nombre	**Region**
Clase metamodelo	Clase
Descripción	Representa cada una de las regiones usadas por las ventanas de interfaz gráfica de usuario de la aplicación Web.
Icono	
Restricciones	-
Valores etiquetados	*DefaultPage* representa la página por defecto asignada a la región.

Tabla 3.8: Estereotipo NMMp Retrieval Function

Nombre	**Retrieval Function**
Clase metamodelo	Asociación
Descripción	Representa una relación dirigida desde un ancla de recuperación a una página estática.
Restricciones	-
Valores etiquetados	-

Tabla 3.9: Estereotipo NMMp Computing Function

Nombre	**Computing Function**
Clase metamodelo	Asociación
Descripción	Representa una relación dirigida desde un ancla computacional a una página dinámica.
Restricciones	-
Valores etiquetados	Input contiene los parámetros necesarios par construir la página dinámica. Este valor etiquetado puede tener formato de cadena y estar codificado.

3.2.3 Transformación NMMp a UML-WAE

Como se ha comentado anteriormente, los modelos NMMp son diseños de alto nivel que, facilitando la comunicación entre desarrolladores y usuarios finales, permiten definir la estructura navegacional de una aplicación Web. Sin embargo, los modelos NMMp están

muy lejos del nivel de implementación donde puede definirse los patrones arquitectónicos y de diseño. Por tanto, su transformación a modelos UML-WAE que expresen implementaciones concretas es esencial en el proceso de desarrollo y mantenimiento de las aplicaciones Web, siempre que no se quiera acoplar el mantenimiento del código a una herramienta propietaria que sea la única forma de traducir diagramas en código.

Como se describió en la sección 2.7, UML-WAE es muy útil para soportar el desarrollo de la capa de presentación de aplicaciones Web reutilizando todos los beneficios de UML, incluyendo la capacidad de generar y mantener los modelos en herramientas UML CASE genéricas y la inclusión explicita de patrones arquitectónicos y de diseño en sus modelos. En UML-WAE todos los contenidos generados por un servidor Web son llamados páginas, las cuales son representadas por clases estereotipadas. UML-WAE define varios tipos de páginas: (i) páginas de cliente, representadas por el estereotipo *Client Page*, que caracterizan páginas Web estructuradas con un lenguaje de marcado como HTML o XML y que son presentadas por los navegadores Web; (ii) páginas de servidor, representadas por el estereotipo *Server Page*, que caracterizan procesos computacionales que se ejecutan en el lado del servidor, para normalmente construir una página de cliente; y (iii) formularios, representados por el estereotipo *Form*, que caracterizan formularios de entrada de datos definidos en las páginas de cliente.

Además, en UML-WAE, las relaciones navegacionales entre páginas se representan por asociaciones navegadas estereotipadas. UML-WAE define varios tipos de relaciones navegacionales: (i) hipervínculos, representados por el estereotipo *Link*, representan enlaces entre dos páginas; (ii) envío de datos entre formularios y procesos computacionales que se ejecutan en el lado del servidor, representados por el estereotipo *Submit*; (iii) reenvío del control entre procesos computacionales que se ejecutan del lado del servidor, representados por el estereotipo *Forward*; y (iv) construcción de páginas de cliente por procesos computacionales que se ejecutan del lado del servidor, representados por el estereotipo *Build*.

Como se ha comentado, en MDA, las actividades más importantes son el modelado de los diferentes aspectos de un sistema y la definición de reglas que permitan transformar modelos abstractos en modelos concretos. Aplicar el enfoque MDA en el contexto de NMMp ha consistido en definir un conjunto de transformaciones explícitas que permitan, a partir de un modelo NMMp del sistema y de la descripción arquitectónica del modelo destino, obtener un modelo UML-WAE con la arquitectura seleccionada de la forma más automatizada posible. En la Tabla 3.10 se describen las reglas de transformación

implementadas durante el desarrollo del presente trabajo para permitir generar automáticamente los modelos UML-WAE con arquitectura Model 1 y Model 2 a partir de los modelos NMMp.

Model 1 (Brown et al., 2002; Brown et al., 2005) es un diseño arquitectónico para aplicaciones Web "*centrado en las páginas*" que utiliza el enfoque cliente/servidor. Este modelo básicamente elimina la presencia de un controlador, permitiendo que las páginas Web del lado del servidor, casi siempre encargadas de tareas de presentación, puedan acceder directamente a las capas intermedias de una aplicación Web para componer la respuesta a una petición de un cliente Web. La página Web del lado del servidor es la encargada de recibir la petición, coordinar el procesamiento, componer y enviar la respuesta al cliente. La ventaja de esta arquitectura es que es simple de implementar. Sin embargo, sólo es adecuada para aplicaciones pequeñas con pocas páginas y baja complejidad.

La arquitectura Model 2 es una implementación del lado del servidor del patrón arquitectónico *Model-View-Controller* (MVC). Este patrón promueve la separación entre el modelo de la aplicación (*Model*) y la forma en la que es presentado (*View*). Esto requiere de un componente separado para coordinar la relación entre ellos (*Controller*). Model 2 utiliza el controlador para manejar todas las peticiones de procesamiento y delegar cada petición en una página Web del lado del servidor diferente, que compone la presentación y responde a la petición, a diferencia de la arquitectura Model 1 en la cual el controlador y la vista coexisten dentro del mismo componente.

Tabla 3.10: Transformación NMMp a UML-WAE Model 1 y Model 2.

Elemento NMMp	**Elemento UML-WAE Model 1**	**Elemento UML-WAE Model 2**
Página estática *p*	Página de cliente *p*	Página de cliente *p*
Página dinámicamente creada *p*	Página de cliente *p* generada por una Página de servidor	Página de cliente *p* generada por una Página de servidor
Ancla de recuperación *a*	-	-
Ancla computacional *a* fuera de un formulario	-	-
Ancla computacional *a* en un formulario dentro de una Página *p*	Formulario *a* agregado a una Página de cliente *p*	Formulario *a* agregado a una Página de cliente *p*

Elemento NMMp	**Elemento UML-WAE Model 1**	**Elemento UML-WAE Model 2**
Enlace desde Ancla de recuperación *a* dentro de una Página *p1* a una Página estática *p2*	Enlace desde Página *p1* a Página de cliente *p2*	Enlace desde Página *p1* a clase *Controller* + dependencia *forward* desde la clase *Controller* a Página de cliente *p2*
Enlace desde Ancla computacional *a* fuera de un formulario dentro de una página *p1* a una Página dinámicamente creada *p2*	Enlace desde la Página *p1* a la Página de servidor *aSP* + operación *a* en la clase Fachada + clase *Application Service aAS* + clases de transferencias de datos *aInputTransfer* y *aOutputTransfer* con dependencias desde la Página de servidor *aSP* y la clase *Application Service aAS* hacia ellas + dependencia *build* desde la Página de servidor *aSP* hasta Página de cliente *p2*	Enlace desde la Página *p1* a clase *Controller* + clase *Action aAction* + operación *a* en la clase Fachada + clase *Application Service aAS* + clases de transferencias de datos *aInputTransfer* y *aOutputTransfer* con dependencias desde la clase *aAction* y la clase *Application Service aAS* hacia ellas + dependencia *forward* desde la clase *Controller* a la página de servidor *aView* + dependencia desde la página de servidor *aView* a la clase de tranferencia de datos *aOutputTransfer* + dependencia *build* desde la Página de servidor *aView* hasta Página de cliente *p2*
Enlace desde Ancla computacional *a* en un formulario dentro de una página *p1* a una Página dinámicamente creada *p2*	Dependencia *submit* desde el Formulario *a* a la Página de servidor *aSP* + operación *a* en la clase Fachada + clase *Application Service aAS* + clases de transferencias de datos *aInputTransfer* y *aOutputTransfer* con dependencias desde la Página de servidor *aSP* y la clase *Application Service aAS* hacia ellas + dependencia *build* desde la Página de servidor *aSP* hasta Página de cliente *p2*	Dependencia *submit* desde el Formulario *a* a clase *Controller* + clase *Action aAction* + operación *a* en la clase Fachada + clase *Application Service aAS* + clases de transferencias de datos *aInputTransfer* y *aOutputTransfer* con dependencias desde la clase *aAction* y la clase *Application Service aAS* hacia ellas + dependencia *forward* desde la clase *Controller* a la página de servidor *aView* + dependencia desde la página de servidor *aView* a la clase de tranferencia de datos *aOutputTransfer* + dependencia

Elemento NMMp	**Elemento UML-WAE Model 1**	**Elemento UML-WAE Model 2**
		build desde la Página de servidor *aView* hasta Página de cliente *p2*
Ventana *w*	Frameset *w*	Frameset *w*
Region *r*	Target *r*	Target *r*
Elemento NMMp	Elemento UML-WAE Model 1	Elemento UML-WAE Model 2
Conexión desde ventana *w* a región *r*	Agregación desde Frameset *w* a Target *r*	Agregación desde Frameset *w* a Target *r*

Como puede observarse en la Tabla 3.10, la transformación de modelos NMMp a modelos UML-WAE implica la generación automática de los componentes del lado del servidor responsables de gestionar el proceso navegacional. Además, estos componentes se relacionan de tal forma que implementan patrones arquitectónicos y de diseño ampliamente usando en la industria. Por ejemplo, los enlaces NMMp son transformados en asociaciones navegadas entre páginas UML-WAE. En Model 1, estas asociaciones se establecen directamente entre páginas, mientras que en Model 2 se establecen por medio de un *Controller* (Fowler, 2002). Las anclas computacionales son transformadas en componentes de capa de negocio capaces de invocar lógica de negocio. Estos componentes pueden ser clases *Fachada* (Gamma et al., 1995) y clases de servicio de aplicación (*Application Service*) (Fowler, 2002) que ejecutan lógica de negocio localmente, o delegados de negocio (*business delegate*) (Fowler, 2002) que representan servicios remotos. El *Controller* invoca estos componentes por medio de *Commands* (Gamma et al., 1995), y el flujo de información entre la capa de lógica de negocio y las páginas se implementa utilizando objetos de transferencia de datos (DTO) (Fowler, 2002) cuya estructura interna dependerá del modelo de datos de la aplicación.

De esta forma, el enfoque NMMp estimula el uso de patrones arquitectónicos y de diseño en el desarrollo de aplicaciones Web, como el sector industrial lo hace, pero, usando al mismo tiempo mapas navegacionales para incrementar el nivel de abstracción del diseño, como promueve el sector académico. De esta forma, con NMMp se logra un equilibrio entre los enfoques usados en la industria y los enfoques promovidos por la academia.

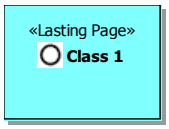
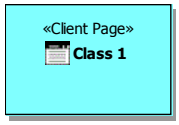
Para ilustrar el proceso de desarrollo con NMMp, hemos implementado el conjunto de reglas de transformación utilizando el lenguaje estándar *QVT-Operational Mappings* (OMG, 2008) implementado por la herramienta CASE Borland Together para

transformar automáticamente los modelos NMMp en modelos UML-WAE que implementan los patrones arquitectónicos Model 1 y Model 2. Las siguientes secciones describen las transformaciones QVT implementadas con base en las transformaciones descritas en la Tabla 3.10. Por razones de concisión, solo se detallan las transformaciones de NMMp a UML-WAE con arquitectura Model 2.

3.2.3.1 Transformación de diagramas de página

Esta sección describe las reglas de transformación QVT implementadas para transformar los diagramas de páginas NMMp en diagramas UML-WAE con arquitectura Model 2. La Figura 3.2 muestra la regla de transformación QVT para transformar una página estática NMMp en una página de cliente UML-WAE. La figura muestra tanto la representación visual de la página NMMp como de la página UML-WAE. La regla de transformación es directa, para cada clase estereotipada con *Lasting Page* en un modelo NMMp de entrada se genera una clase estereotipada con *Client Page* en el modelo UML-WAE de salida. Con la cláusula "*When*" se asegura que la regla será aplicada solo a clases estereotipadas con *Lasting Page*, la nueva clase producida conservará el mismo nombre y descripción.

Figura 3.2: Transformación de página estática NMMp a UML-WAE

NMMp lasting page	Páginas UML WAE
	

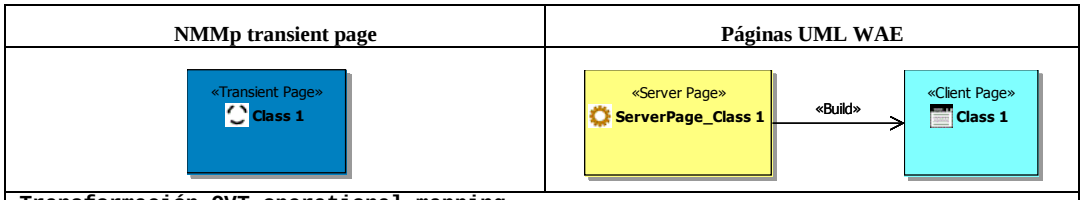
Transformación QVT operational mapping

```
mapping uml20::classes::Class::ToClientPage() : uml20::classes::Class
        when
        {
                self.stereotypes = OrderedSet{'Lasting Page'}
        }
        {
          object
          {
                name := self.name;
                description := self.description;
                stereotypes := OrderedSet { 'Client Page' };
          }
        }
```

La Figura 3.3 muestra las reglas de transformación QVT para transformar una página dinámica NMMp en las correspondientes páginas UML-WAE. La figura muestra tanto la representación visual de la página NMMp como de las páginas UML-WAE. Como se puede ver, en este caso se aplican tres reglas de transformación. En la primera, la clase *Transient Page* de NMMp se transforma en la clase *Client Page* de UML-WAE

conservando su mismo nombre y descripción. Con la segunda regla, la misma clase *Transient Page* se transforma en la clase *Server Page* de UML-WAE, aquí por razones de legibilidad se le antepone la palabra *ServerPage* al nombre. Y con la tercera regla se construye la asociación estereotipada con *Build* entre las clases construidas anteriormente.

Figura 3.3: Transformación de página dinámica NMMp a UML-WAE



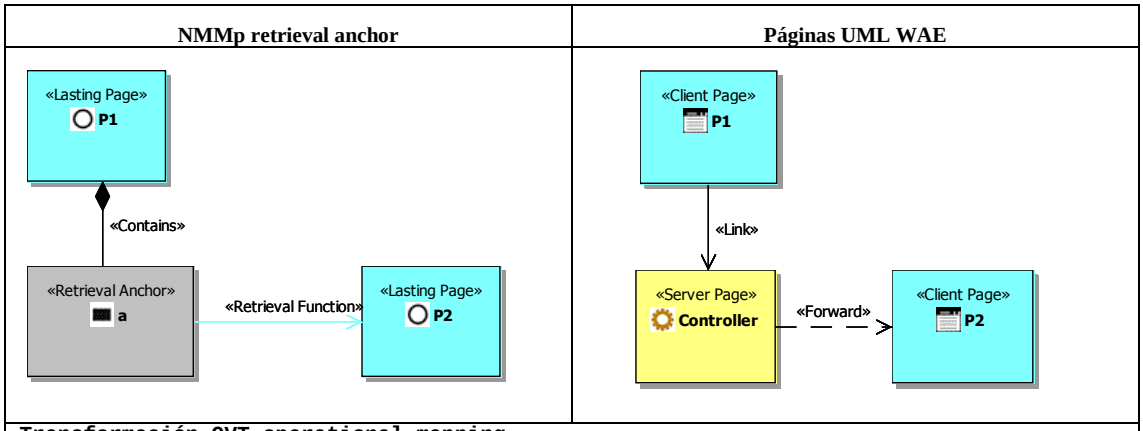
Transformación QVT operational mapping

```
mapping uml20::classes::Class::ToClientPage() : uml20::classes::Class
        when
        {
                self.stereotypes = OrderedSet{'Transient Page'}
        }
        {
          object
          {
                name := self.name;
                description := self.description;
                stereotypes := OrderedSet { 'Client Page' };
          }
        }

mapping uml20::classes::Class::ToServerPage() : uml20::classes::Class
        when
        {
                self.stereotypes = OrderedSet{'Transient Page'}
        }
        {
          object
          {
                name := 'ServerPage_' + self.name;
                description := self.description;
                stereotypes := OrderedSet { 'Server Page' };
          }
        }

mapping uml20::classes::Class::ToBuild() : uml20::classes::Dependency
        {
          object
          {
                client      := self.resolve(uml20::classes::Class)->
                                        select(c | c.stereotypes->
                                      includes('Server Page'));
                supplier    := self.resolve(uml20::classes::Class)->
                                        select(c | c.stereotypes->
                                  includes('Client Page'));
                stereotypes := OrderedSet { 'Build' };
          }
        }
```

La Figura 3.4 muestra las reglas de transformación QVT para transformar un ancla de recuperación que enlaza dos páginas estáticas NMMp en las correspondientes páginas UML-WAE. La figura muestra tanto la representación visual de las páginas NMMp como de las páginas UML-WAE. Como se puede ver, en este caso se aplican tres reglas de

transformación. En la primera, se comprueba si la clase UML-WAE estereotipada con *Controller* ha sido creada previamente, en caso negativo se procede a crearla a través de la segunda regla de transformación. Con la tercera regla de transformación se crea la asociación estereotipada con *Forward*.

Figura 3.4: Transformación de ancla de recuperación NMMp a UML-WAE



Transformación QVT operational mapping

```
query uml20::classes::Class::HasBeenConvertedToController() : Boolean
    {
         if((self.resolve(uml20::classes::Class)->select(class |
             class.name = 'Controller'))->any() = true) then

             true

         else

             false

         endif
    }

mapping uml::together::Model::ToControllerClass() : uml20::classes::Class
    {
         object
         {
               name := 'Controller';
               stereotypes := OrderedSet{ 'Server Page' };
         }
    }

mapping uml20::classes::Dependency::ToForwardDependencies(in NMMModel :
uml::together::Model) : uml20::classes::Dependency
    {
         init
         {
      var Supplier := self.supplier.oclAsType(uml20::classes::Class);

         result := NMMModel.CreateDependency(Supplier,'Controller', 'Client Page',
'Forward');
         }
}
```

La Figura 3.5 y 3.6 muestran las reglas de transformación QVT para transformar en las correspondientes páginas UML-WAE un ancla computacional en un formulario que enlaza una página estática con una página dinámica NMMp. La Figura 3.5 muestra tanto la representación visual de las páginas NMMp como de las páginas UML-WAE. Como

se comentó anteriormente, la presencia de anclas computacionales en los modelos NMMp implica crear la capa de lógica de negocio en el modelo UML-WAE con arquitectura Model 2. La clase estereotipada con *Form Computing Anchor* en primera instancia, debe ser transformada a una clase estereotipada con *Form* que en UML-WAE representa un conjunto de campos de entrada con la capacidad de enviar *(Submit)* esta información al servidor Web, lanzando un proceso computacional. Es necesario también comprobar si la clase *Controller* y la interfaz *action* han sido creadas previamente en la capa de lógica de negocio y en caso negativo proceder a crearlas.

La lógica de negocio es implementada por una clase estereotipada con *Application Service,* que se desacopla de la clase *Controller* a través de la clase estereotipada con *Action* que implementa el interfaz *action*. Por tanto, la clase estereotipada con *Form Computing Anchor* debe ser transformada también en dos clases con estos estereotipos. También es necesario agregar un nuevo método en la clase *Fachada* que represente el nuevo proceso computacional, crear las clases estereotipadas con *DTO Input* y *DTO Output* para modelar el flujo de información entre la capa de negocio y las clases estereotipadas con *Server Page*, y por último generar las relaciones de dependencia adecuadamente.

La Figura 3.6 muestra las tres reglas de transformación QVT que permiten las transformaciones descritas. En la primera, se crea el interfaz *action* en caso de no haber sido creada antes. Con la segunda regla, a partir de la clase estereotipada con *Form Computing Anchor* se crean las clases UML-WAE estereotipadas con *Server Page*, *Action*, *Application Service*, *DTO Input* y *DTO Output* de la capa de negocio de la arquitectura Model 2. Y con la tercera regla se generan las dependencias adecuadas entre las clases anteriormente creadas.

Figura 3.5: Transformación de ancla computacional en formulario NMMp a UML-WAE

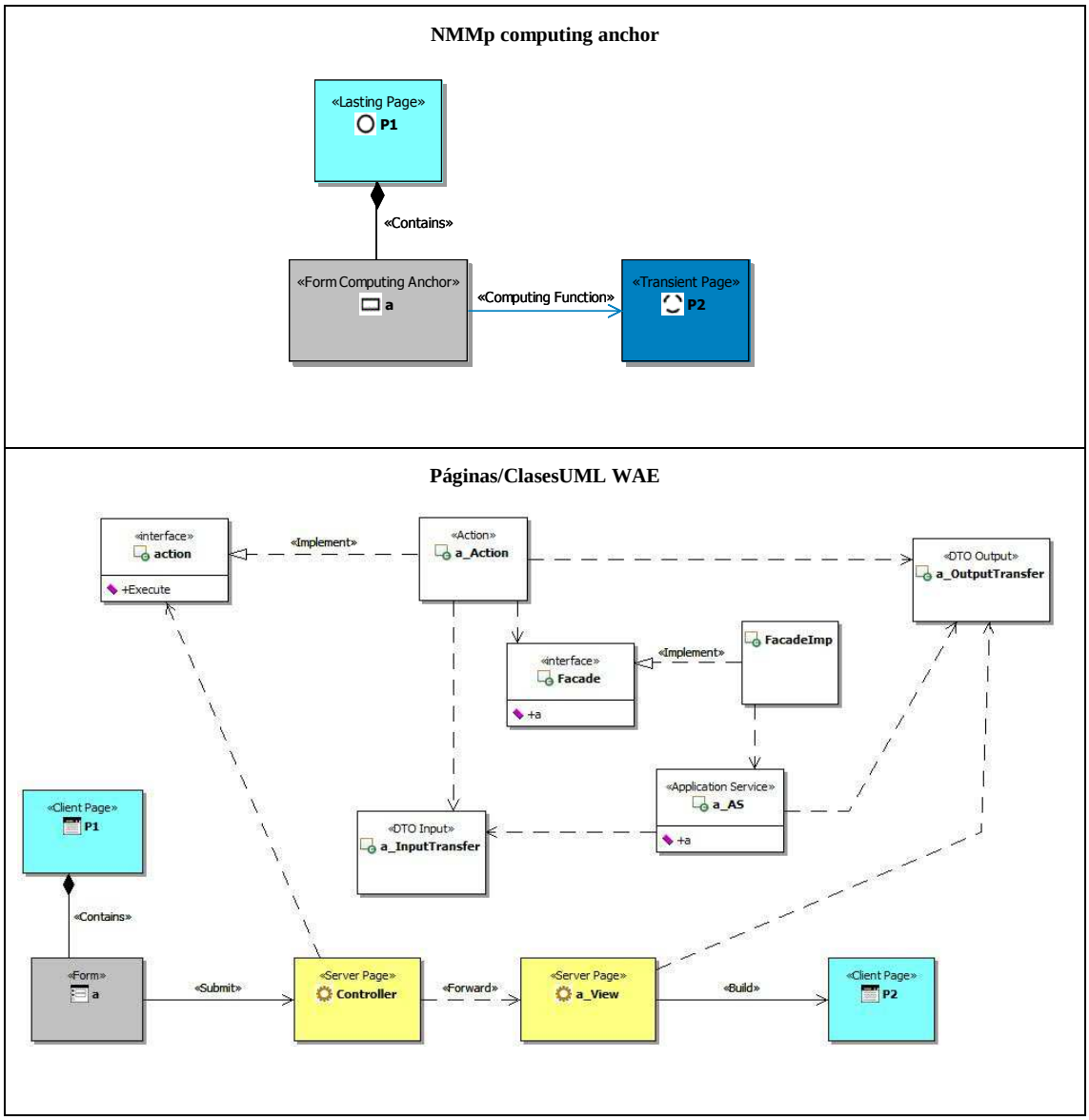


Figura 3.6: Transformación QVT de ancla computacional en formulario NMMp a UML-WAE

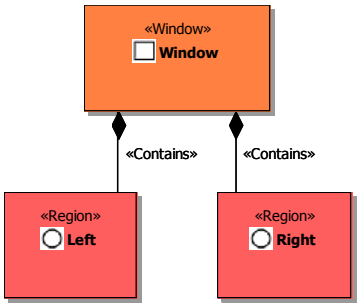
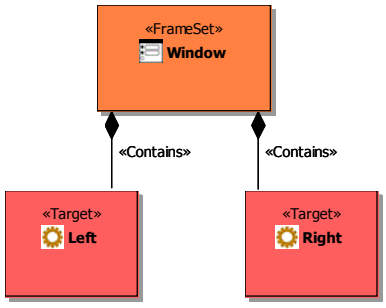
Transformación QVT operational mapping

```
mapping uml::together::Model::ToActionClass() : uml20::classes::Class
{
         object
         {
            name := 'action';
            stereotypes := OrderedSet{ 'interface' };
            ownedOperations += object uml20::kernel::features::Operation
                                                { name := 'Execute'; }
         }
}

query uml20::classes::Class::CreateM2BussinessLayerClasses() :
Set(uml20::classes::Class)
{
         Set
         {
             self.ToWAEClass(self.name + '_View', 'Server Page', ''),
             self.ToWAEClass(self.name + '_Action', 'Action', ''),
             self.ToDTOClass(self.name + '_InputTransfer', 'DTO Input'),
             self.ToDTOClass(self.name + '_OutputTransfer', 'DTO Output'),
             self.ToApplicationService()
         }
}

query uml20::classes::Class::CreateM2Dependencies(in NMMModel : uml::together::Model)
: Set(uml20::classes::Dependency)
{
         Set
         {
            NMMModel.CreateDependency(self, 'FacadeImp',
                                      'Application Service'),
            NMMModel.CreateDependency(self, 'Controller',
                                      'Server Page', 'Forward'),
            self.CreateDependency(NMMModel, 'Action', 'Facade'),
            self.CreateDependency(NMMModel, 'Action', 'action',
                                              'Implement'),
            self.ToDTOOutputDependencies()
         }
}
```

3.2.3.2 Transformación de diagramas de región

Esta sección describe las reglas de transformación QVT implementadas para transformar los diagramas de región NMMp en diagramas UML-WAE. La Figura 3.7 describe la regla de transformación para transformar clases NMMp estereotipadas con *Windows* y *Regions* en componentes UML-WAE. Básicamente, las clases estereotipadas con *Windows* y *Regions* se transforman en clases UML-WAE estereotipadas con *Framesets* y *Targets* conservando las asociaciones de composición. Por supuesto, las reglas de transformación pueden ser modificadas levemente para generar otros elementos HTML como regiones construidas con elementos Div.

Figura 3.7: Transformación diagramas de regiones NMMp a UML-WAE

Elementos diagrama de región NMMp	Componentes UML WAE
	

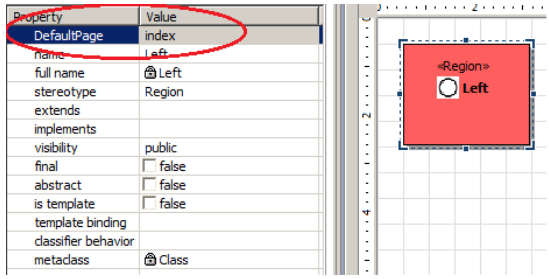
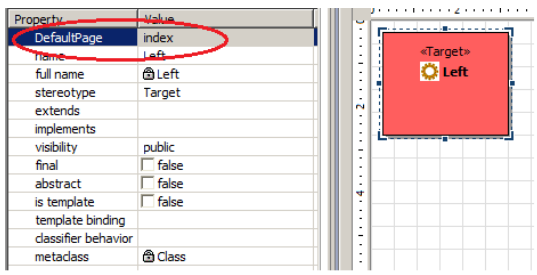
Transformación QVT operational mapping

```
mapping uml20::classes::Class::ToFrameset() : uml20::classes::Class
        when
        {
          self.stereotypes = OrderedSet{Window'}
        }
        {
          object
          {
                name := self.name;
                description := self.description;
                stereotypes := OrderedSet { 'FrameSet' };
          }
        }

mapping uml20::classes::Class::ToTarget() : uml20::classes::Class
        when
        {
          self.stereotypes = OrderedSet{Region}
        }
        {
          object
          {
                name := self.name;
                description := self.description;
                stereotypes := OrderedSet { 'Target' };
          }
        }

mapping uml20::classes::Class::ToTarget(inout Target
uml20::classes::Class):uml20::classes::Class
        when { self.stereotypes = OrderedSet{'Region'} }
        {       init
          {     Target.name := self.name;
                Target.description := self.description;
                Target.stereotypes := OrderedSet { 'Target' };
                Target.setPropertyValue('DefaultPage',
        self.getPropertyValue('DefaultPage'));
                result := Target;
          }
        }

mapping uml20::classes::Class::ToLink(inout Link : uml20::classes::Dependency) :
uml20::classes::Dependency
        { init
                { Link.client := self.dependencies->select(d | d.stereotypes->
        includes('Contains'))-> collect(dc |
        dc.client.oclAsType(uml20::classes::Class))
                  Link.supplier := self.dependencies->select(d |d.stereotypes-
        >includes('Retrieval
                                              Function'))->collect(dc |
                           c.supplier.oclAsType(uml20::classes::Class));
                Link.stereotypes := OrderedSet{ 'Link' };
                Link.setPropertyValue('Target',self.getPropertyValue('Target'));
        }
}
```

3.2.3.3 Transformación de diagramas de mezcla

Cada elemento *Region* de un modelo NMMp posee un atributo llamado *DefaultPage* que contiene el nombre de la página por defecto asignada a cada región. Igualmente, se ha definido en el perfil UML-WAE el valor etiquetado *DefaultPage* asociado a las clases estereotipadas con *Target*, de forma que todos estos elementos poseen un atributo llamado *DefaultPage* que contiene el nombre de la página por defecto asignada. Por tanto, la transformación de la función de asignación de página por defecto a los elementos *Region* en NMMp se hace simplemente copiando el valor contenido en el atributo *DefaultPage* de estos elementos al atributo *DefaultPage* de los elementos *Target* de UML-WAE. La Figura 3.8 muestra la representación visual de la función de asignación de página por defecto a las clases estereotipadas con Region de NMMp y a las clases estereotipadas con Target de UML-WAE así como la regla de transformación QVT.

Figura 3.8: Transformación páginas por defecto NMMp a UML-WAE

NMMp Default page	UML WAE Default page
	

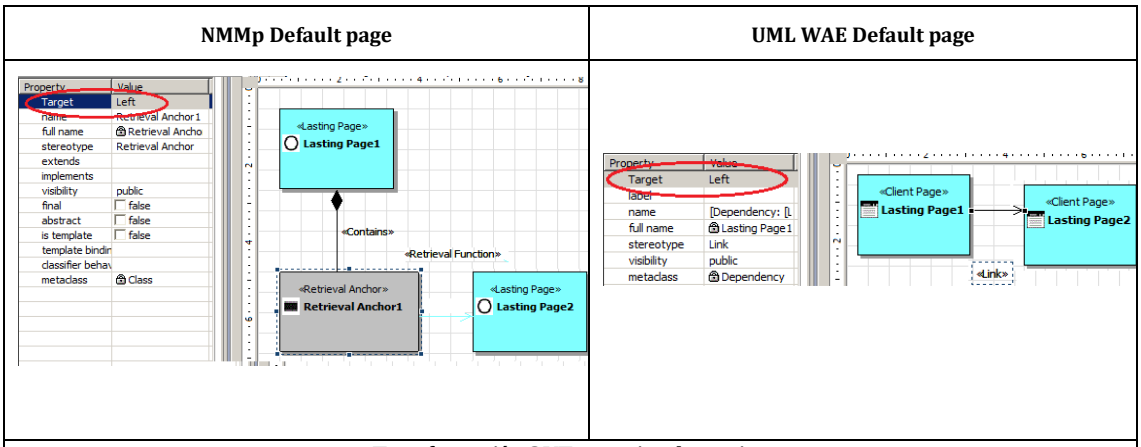
Transformación QVT operational mapping

```
mapping uml20::classes::Class::ToTarget(inout Target :
                          uml20::classes::Class) : uml20::classes::Class
when
{
         self.stereotypes = OrderedSet{'Region'}
}
{
         init
         {
               Target.name := self.name;
               Target.description := self.description;
               Target.stereotypes := OrderedSet { 'Target' };
              Target.setPropertyValue('DefaultPage',
                         self.getPropertyValue('DefaultPage'));
         result := Target;
         }
}
```

De forma semejante, la función de asignación de la región destino a las anclas NMMp se ha implementado asociando un valor etiquetado llamado *Target* a las clases estereotipadas con *Retrieval Anchor*, *Form Computing Anchor* y *Non Form Computing Anchor*. Por tanto, cada uno de estos elementos en un modelo NMMp posee un atributo

llamado *Target* que contiene el nombre de la región por defecto asignada a cada ancla. De igual forma, en el perfil UML-WAE se ha asociado el valor etiquetado *Target* a las clases estereotipadas con *Link* y *Submit*, de forma que estos elementos posean el atributo *Target* en los modelos UML-WAE. La transformación de la función de asignación de la región destino de las anclas NMM se efectúa copiando el valor del atributo *Target* de las anclas NMM a los elementos *Link* o *Submit* de los modelos UML-WAE. La Figura 3.9 muestra la representación visual de la función de asignación de la región destino a las anclas NMMp y a los elementos *Link* o *Submit* de UML-WAE así como la regla de transformación QVT.

Figura 3.9: Transformación regiones destino NMMp a UML-WAE



Transformación QVT operational mapping

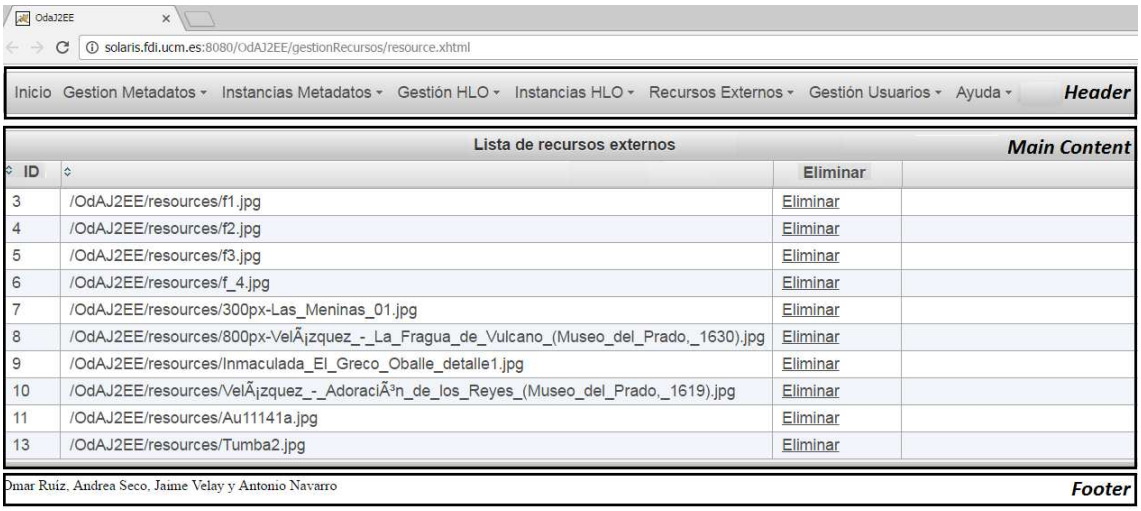
```
mapping uml20::classes::Class::ToLink(inout Link :
uml20::classes::Dependency) : uml20::classes::Dependency
{
  init
  {
     Link.client := self.dependencies->
                    select(d | d.stereotypes->
                    includes('Contains'))->
                    collect(dc |
                        dc.client.oclAsType(uml20::classes::Class))
      Link.supplier := self.dependencies->select(d |
                       d.stereotypes->includes('Retrieval
                                                Function'))->
                       collect(dc |
      dc.supplier.oclAsType(uml20::classes::Class));
     Link.stereotypes := OrderedSet{ 'Link' };
     Link.setPropertyValue('Target',
                        self.getPropertyValue('Target'));
  }
}
```

3.3 Ejemplo de aplicación de NMMp: OdAJ2EE

En la sección anterior se ha definido la notación NMMp. En esta sección se mostrará un ejemplo real de su aplicación. Para facilitar la ilustración del proceso, se comentará un caso de uso de una aplicación Web desplegada en un entorno real, después se modelará haciendo uso del perfil NMMp, para finalmente obtener, por medio de la ejecución automática de las reglas QVT, el modelo UML-WAE correspondiente.

OdAJ2EE[7] es una aplicación Web desarrollada en Java para administrar objetos de aprendizaje híbridos, los cuales usan una estructura de meta-datos específica de un dominio concreto dinámicamente definida (Navarro et al., 2013). Un caso de uso en OdAJ2EE es la eliminación de entidades *Recurso* gestionadas por la aplicación, el cual sólo pueden ejecutar administradores de la aplicación. La Figura 3.10 muestra una captura de pantalla de la aplicación implementando este caso de uso. Como se puede ver, la estructura de la aplicación está basada en una plantilla con tres regiones llamadas: *Header, Main Content y Footer*. La página actual muestra una tabla de recursos con un enlace en cada uno de ellos que proporciona los medios para eliminarlo. Si un administrador selecciona uno de ellos, la aplicación navega a una página de confirmación que muestra los detalles del recurso seleccionado y pide confirmación para eliminarlo. Si el administrador confirma la eliminación del recurso, la aplicación intenta eliminarlo mostrando una página informativa en caso de éxito o una página de error en caso contrario.

Figura 3.10: Eliminación de recursos en OdAJ2EE



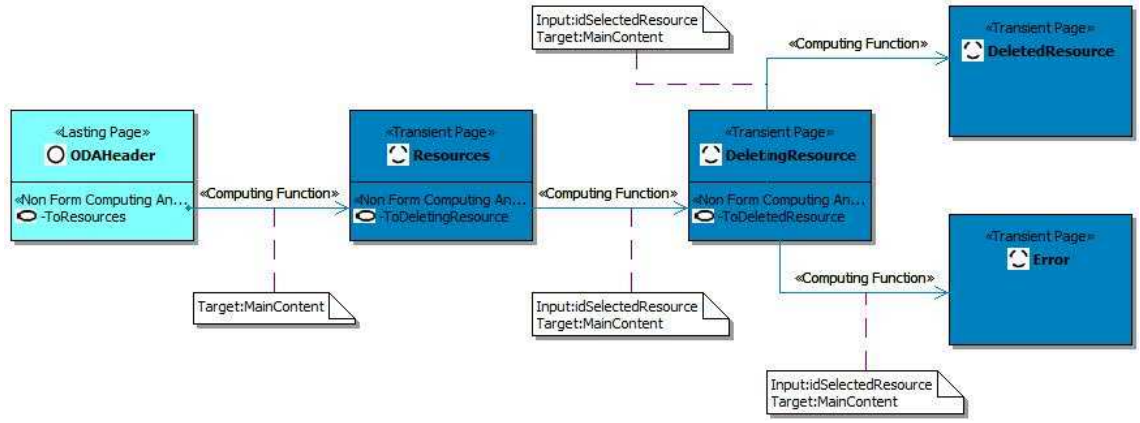
3.3.1 OdAJ2EE con NMMp

La Figura 3.11 muestra el modelo NMMp que caracteriza la estructura navegacional del caso de uso anterior. Como puede observarse, la página *ODAHeader* (la cual se carga en la región *Header*) incluye un ancla computacional fuera de formulario llamada *toResources*, que da acceso a la página *Resources*. Esta página muestra la tabla de

[7] http://solaris.fdi.ucm.es:8080/OdAJ2EE/

entidades *Recurso* disponibles en la aplicación con un enlace que proporciona los medios para eliminarlos. Este enlace es caracterizado por el ancla computacional fuera de formulario llamada *toDeletingResource* que da acceso a la página *DeletingResource* enviándole el identificador del recurso seleccionado. Esta página es responsable de mostrar los detalles del recurso y ejecutar el proceso de eliminación. Este proceso puede ser satisfactorio navegando a la página *DeletedResource* o puede fallar navegando a la página *Error* a través del ancla computacional fuera de formulario llamada *toDeleteResource*. Nótese como las asociaciones *Computing Function* transportan la información entre páginas y establecen la región en la cual las páginas destino serán mostradas en sus valores etiquetados, los cuales, por claridad, se muestran como notas UML.

Figura 3.11: Eliminación de recursos en OdAJ2EE con NMMp



Como puede verse en la Figura 3.11, los diagramas NMMp son diagramas abstractos, independientes de cualquier arquitectura software y lenguaje de programación, que no se ocupan de modelar los componentes funcionales del lado del servidor. El modelo de páginas NMMp sólo se ocupa de modelar la capa de presentación Web, describiendo las páginas y su relación navegacional, haciendo que los modelos sean sencillos y muestren un visión general de la aplicación, y facilitando, por tanto, la comunicación entre desarrolladores y usuarios finales. Cabe destacar que los modelos NMMp pueden ser generados y mantenidos en herramientas UML CASE genéricas.

3.3.2 OdAJ2EE con UML-WAE

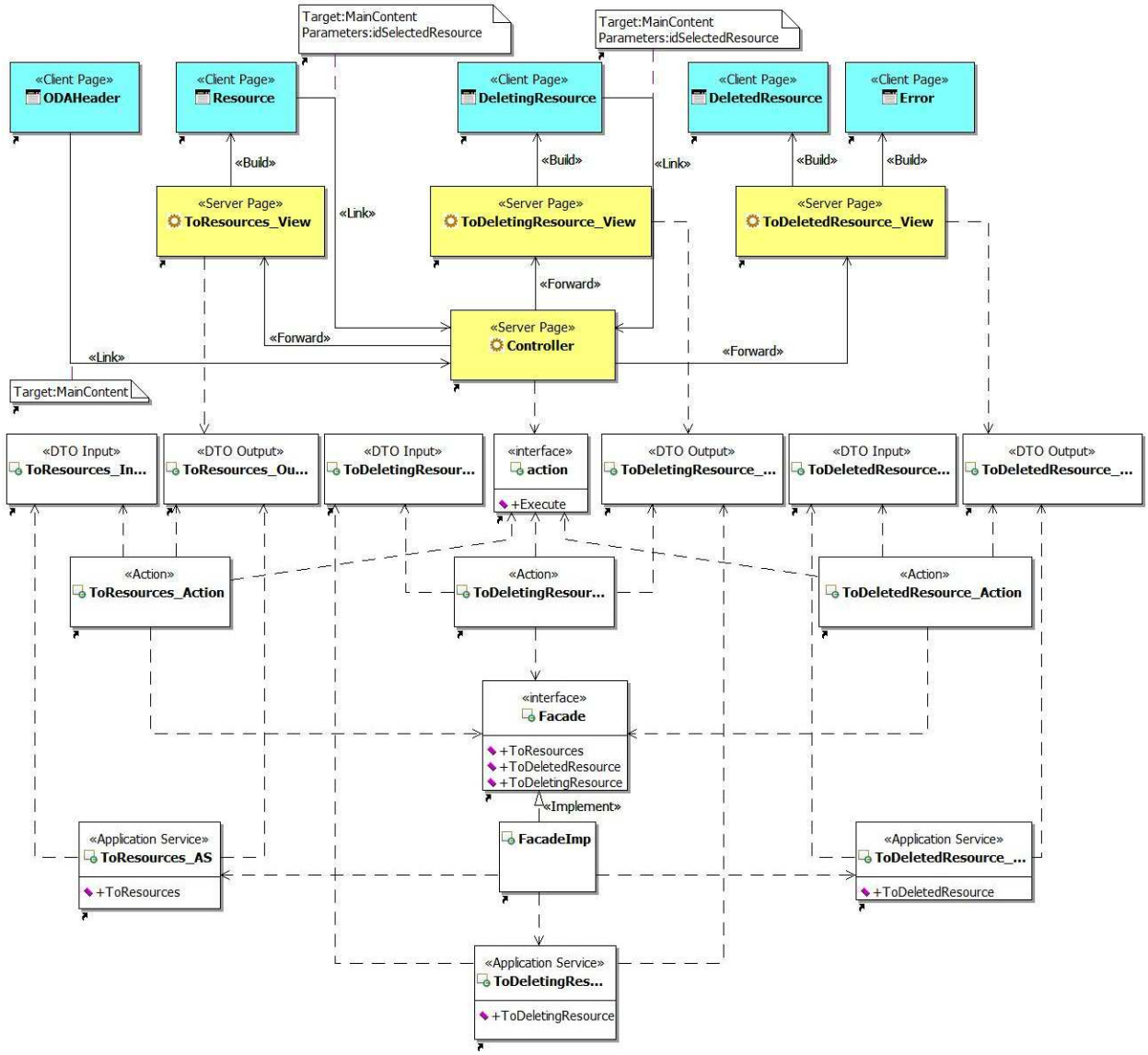
A partir del modelo NMMp desarrollado en la sección anterior (ver Figura 3.11), y por medio de la aplicación de las reglas de transformación NMMp a UML-WAE, se obtiene el modelo UML-WAE con arquitectura Model 2 mostrado en la Figura 3.12.

Como se puede ver en las Figuras 3.11 y 3.12, las páginas NMMp han sido convertidas en páginas UML-WAE, y las anclas computacionales han sido utilizadas para crear todos los componentes computacionales de la capa de negocio. De esta forma, las anclas computacionales NMMp ocultan las acciones invocadas por las páginas de servidor, la clase *Fachada* responsable de la implementación de las funcionalidades (usando clases especializadas estereotipadas con *Application Service*) y los objetos de transferencia de datos que mueven los datos entre la capa de negocio y la capa de presentación. Los enlaces entre anclas y páginas en el diagrama NMMp han sido transformados en enlaces entre los elementos UML-WAE.

Como se muestra en la Figura 3.12, UML-WAE se aplica sólo a la capa de presentación de las aplicaciones Web, sin embargo el flujo navegacional mostrado claramente por el diagrama NMMp de la Figura 3.11 se vuelve confuso en la Figura 3.12. En general los diagramas NMMp son valiosos tanto para usuarios como para desarrolladores, mientras que los diagramas UML-WAE son valiosos solo para los desarrolladores, quienes son los responsables de implementar la estructura navegacional y los artefactos computacionales necesarios para construir la aplicación Web. Es importante señalar que mientras el diagrama NMMp de la Figura 3.11 tiene solo ocho clases UML (cinco páginas y tres anclas computacionales), el diagrama UML-WAE de la Figura 3.12 tiene veinticuatro clases. Por tanto, es evidente el poder descriptivo de los elementos NMMp.

Debido a que el diagrama UML-WAE de la Figura 3.12 es UML, puede ser integrado directamente con otros diagramas UML que representen el resto de capas de la aplicación, usando cualquier herramienta UML CASE genérica. Como puede verse, algunos patrones multicapas son generados en la transformación (*Application Services, Controllers, Data Transfer Objects*, etc.) y otros pueden ser explícitamente agregados para implementar soluciones específicas de negocio. De esta forma, el enfoque NMMp estimula el uso de patrones arquitectónicos y de diseño en el desarrollo de aplicaciones Web, como promueve la industria, pero, usando al mismo tiempo mapas navegacionales para incrementar el nivel de abstracción, como promueve la academia. De esta forma, con NMMp se logra un equilibrio entre los dos sectores.

Figura 3.12: Eliminación de recursos en OdAJ2EE con UML-WAE Model 2



4 WAE4JSF – WAE4.NET

4.1 Introducción

Como se comentó en el Capítulo 3, NMMp permite combinar las ventajas de las notaciones de alto nivel promovidas por las metodologías de Ingeniería Web con la implementación de patrones arquitectónicos y de diseño en herramientas UML CASE genéricas. A diferencia de las aproximaciones analizadas en este trabajo, en NMMp las transformaciones a UML-WAE son reglas de transformación explícitas que pueden ser aplicadas manual o automáticamente eliminando la necesidad de herramientas propietarias (*black-box wizards*).

Otra diferencia con las aproximaciones analizadas es que NMMp ha sido desarrollada con el concepto de la arquitectura multicapa (*mutitier architecture*) en mente, la cual ha demostrado ser la arquitectura más notable en el desarrollo de las aplicaciones empresariales (Alur et al., 2003; Fowler 2002; Crawford & Kaplan 2003). A pesar de que NMMp permite modelar aplicaciones Web con arquitecturas concretas, la falta de soporte a frameworks industriales dificulta su aplicación en el desarrollo de aplicaciones empresariales modernas. Además, NMMp no provee mecanismos automáticos de generación de código. Esta es una desventaja de NMMp respecto a las aproximaciones analizadas, las cuales generalmente permiten generar el código de las aplicaciones listo para ser desplegado (aunque la generación de código dependa íntegramente de un generador de código propietario).

El desarrollo de aplicaciones empresariales modernas incluye una alta variedad de plataformas, entre las cuales, J2EE y Microsoft .NET son dos de las más importantes (Thomas et al., 2015). Esta afirmación concuerda con un informe de la agencia *WinterGreen Research* que asegura que la plataforma J2EE soporta el 60% de las

aplicaciones empresariales desplegadas en la Web (WinterGreen, 2014), y con los informes de Microsoft que aseguran que su plataforma .NET tuvo más de seis millones de usuarios en el año 2014 (Microsoft, 2014).

Estas plataformas, en línea con los principales catálogos de patrones arquitectónicos y de diseño, están estructuradas de acuerdo a la arquitectura multicapa. Por ejemplo, para el desarrollo de la capa de presentación, ofrecen los frameworks JavaServer Faces (JSF) (Geary & Horstmann, 2010) y ASP.NET MVC (Esposito, 2011) entre otros. Para el desarrollo de servicios que implementan la capa de negocio, ofrecen los frameworks Java API for XML Web Services (JAX-WS) (Hansen, 2007) y Windows Communication Foundation (WCF) (Lowy, 2010) entre otros. Finalmente, para el desarrollo de la capa de persistencia de datos, ofrecen los frameworks Java Persistence API (JPA) (Keith & Schincariol, 2013) y ADO.NET Entity Framework (EF) (Lerman, 2012) entre otros.

UML sigue siendo suficiente para modelar las capas de negocio e integración de las aplicaciones desarrolladas con estas plataformas. UML permite incluir en sus modelos todo el catálogo de patrones arquitectónicos y de diseño que, hasta cierto punto, soportan la implementación de los frameworks mencionados. Así, UML permite el modelado de *application services* para implementar los servicios de capa de negocio, *transfer objects* para mover datos entre capas, *data access objects* o *domain store* para la persistencia de objetos de negocio, *web services broker* para la publicación de servicios en una arquitectura orientada a servicios (SOA) (Erl, 2007) o *service activator* para la publicación de servicios en una arquitectura dirigida por eventos (EDA) (Etzion & Niblett, 2010). Además, existen perfiles UML que caracterizan frameworks concretos de estas capas, por ejemplo, la herramienta IBM Rational Software Architect 9.x incluye perfiles para frameworks como JPA, que permiten la generación de código Java a partir de diagramas UML.

Sin embargo, ninguna de las aproximaciones analizadas, ni NMMp, permiten incluir en sus modelos componentes de la capa de presentación Web implementados en plataformas específicas, como J2EE o Microsoft.NET. De ahí que en las aproximaciones analizadas se requieran herramientas específicas para generar el código de la capa de presentación Web de aplicaciones empresariales (*black-box wizards*). Por su parte, en NMMp los modelos generados no pueden transformarse directamente en código que implemente la capa de presentación Web con frameworks específicos como los ofrecidos por las plataformas J2EE o Microsoft .NET.

Para salvar este obstáculo, el presente trabajo extiende la definición de UML-WAE para permitirle modelar la capa de presentación Web de aplicaciones empresariales desarrolladas con los frameworks JSF y ASP.NET MVC. Siguiendo la filosofía de UML-WAE, nuestras extensiones definen perfiles UML específicos que hemos llamado WAE4JSF y WAE4.NET (abreviado, WAE4x). Por tanto, los mapas navegacionales desarrollados con estos perfiles son modelos específicos de la plataforma (PSM) directamente traducibles a código sin necesidad de herramientas específicas a las que vincular su mantenimiento.

La generación automática de código es una característica muy valiosa en el proceso de desarrollo de software. De hecho, el enfoque presentado en este trabajo tiene como objetivo proporcionar esta característica. Sin embargo, el presente trabajo sigue los principios del enfoque MDA, en los cuales los modelos PIM deben ser transformados en modelos PSM antes de la generación de código. Los modelos PSM son modelos que representan código, y por tanto, actúan como modelos puente entre los abstractos modelos PIM y el código ejecutable. Debido a que los perfiles WAE4x son meta-modelos para plataformas específicas, estos, al igual que UML, permiten modelar código directamente, usando herramientas UML CASE genéricas, sin necesidad de aprender una nueva notación de diseño, sin la limitación de un conjunto predefinido de primitivas de modelado centradas en el dominio Web, y con todo el poder que los catálogos de patrones arquitectónicos y de diseño proporcionan. Por supuesto, se requiere conocimientos de los frameworks JSF o ASP.NET MVC, pero esto es fundamental en un diseñador que esté desarrollando aplicaciones basadas en estos frameworks.

Debido a que los modelos WAE4x son modelos PSM basados en UML que pueden ser directamente transformados en código, la generación y mantenimiento de éste no depende de herramientas específicas. Ésta es una característica clave que no es resuelta en ninguna de las aproximaciones analizadas en este trabajo. Ninguna de ellas permite a los desarrolladores tener control completo sobre la generación de código, por tanto, es muy difícil entender y modificar directamente el código generado. Y si el código se modifica de forma manual, los modelos se convierten en modelos obsoletos que sólo pueden ser actualizados usando su herramienta específica. Esta es una característica de todas las aproximaciones que no definen modelos PSMs basados en UML.

Como en el caso de UML-WAE, WAE4x se centran en el aspecto navegacional de la capa de presentación Web. Además los frameworks JSF y ASP.NET MVC comparten características comunes, como la definición de la interfaz de usuario a través de plantillas,

y el control de acceso basado en roles (RBAC). Para aprovechar estas capacidades, los perfiles definen componentes que permiten caracterizar estas funcionalidades y generar el código que las implementa de forma transparente.

El modelado de la estructura de la interfaz de usuario (más allá de la definición de plantillas) como por ejemplo, campos de texto, etiquetas, botones, etc. es directo usando UML, ya que estos componentes son clases orientadas a objetos que pueden ser directamente incluidas en los diagramas UML de clases y de secuencia. En nuestra opinión, los otros detalles de la interfaz de usuario no requieren ser incluidos en los modelos UML porque tienden a ser más artísticos y menos relacionados con la implementación de reglas de negocio (Conallen, 1999).

Respecto a las características de seguridad, estas, usualmente, son incluidas al final del proceso de desarrollo de las aplicaciones Web, lo cual a menudo genera innumerables problemas de integración, por lo cual, es altamente recomendable integrarlas en la etapa de diseño de las aplicaciones (Meier, 2006; Steel et al., 2005). Las plataformas J2EE y Microsoft .NET proporcionan soporte para el control de acceso basado en roles (RBAC) de forma declarativa a nivel de páginas y acciones respectivamente. El soporte declarativo para RBAC permite separar las características de seguridad de la lógica de negocio de las aplicaciones y asegurar que estas son implementadas consistentemente a lo largo de toda la aplicación.

Para sacar ventaja de estas capacidades, los perfiles WAE4x definen estereotipos que permiten modelar características de seguridad asignando el control de acceso basado en roles (RBAC) a nivel de páginas y acciones respectivamente. De esta forma, las características RBAC representadas en los modelos WAE4x se usan para generar los componentes específicos que las soportan en las plataformas J2EE y Microsoft .NET.

Por otro lado, J2EE ofrece varios frameworks para implementar la capa de presentación Web, entre los cuales, JSF y Spring MVC son dos de los más reconocidos (Pronschinske, 2015). En este trabajo hemos decidido caracterizar JSF porque es el framework promovido por Oracle, el promotor oficial de J2EE. Hemos tomado una decisión similar respecto a la plataforma Microsoft .NET, la cual ofrece dos frameworks para implementar la capa de presentación Web, el clásico ASP.NET y ASP.NET MVC. El primero ha sido diseñado sobre el patrón de diseño *Page-Controller* (Fowler, 2002) el cual encaja bien en el desarrollo de aplicaciones Web pequeñas o de baja complejidad, donde prueba y mantenibilidad no son aspectos clave (Esposito, 2011). Por tanto, el presente trabajo ha

seleccionado el framework ASP.NET MVC el cual promueve el principio de separación de problemas (*Separation of Concerns*, SoC) y la arquitectura multicapa.

Los frameworks JSF y ASP.NET MVC han sido diseñados para soportar el desarrollo de la capa de presentación Web de aplicaciones empresariales modernas. Por tanto, implementan el patrón arquitectónico MVC (Fowler, 2002), y permiten la inclusión de todo el catálogo de patrones arquitectónicos y de diseño en las capas de negocio e integración. De esta forma se puede usar WAE4x para modelar la capa de presentación Web, y UML para modelar el resto de capas, permitendo el modelado de una aplicación empresarial completa aprovechando todos los beneficios de UML y habilitando la generación de código desde los modelos de forma transparente.

Como se comentó, el presente trabajo especifica reglas explícitas para transformar los modelos WAE4x en código. Estas reglas pueden ser implementadas en motores de transformación para soportar la transformación automática de modelos a código, dando total control sobre el código generado. En este trabajo hemos implementado estas reglas de transformación conforme al motor de transformación modelo a texto (*model-to-text*, M2T) proporcionado por la herramienta CASE Borland Together 2008 (ahora parte de MicroFocus[8]). Así, se dispone de un entorno integrado para la definición de perfiles UML junto con las capacidades de transformación de los modelos a texto.

De esta forma, los modelos PSM producidos usando los perfiles WAE4x se pueden transformar automaticamente, por medio del conjunto de reglas M2T, en un fichero XML intermedio que describe los ficheros de código y de configuración involucrados en la lógica navegacional, además de la estructura de directorios que cada plataforma impone o promueve. Desde el punto de vista de este trabajo, este enfoque es el más natural, debido a que la mayoría de código que implementa la capa de presentación Web con los frameworks JSF y ASP.NET MVC se desarrolla en formato XML (por ejemplo, todas las páginas Web y los ficheros de configuración).

El fichero XML intermedio describe toda la estructura de directorios (que se corresponde con la estructura de paquetes UML), los ficheros contenidos en ellos (que se corresponden con la mayoría de clases UML), y el contenido de estos ficheros (que se corresponden con los atributos y asociaciones UML entre clases). De forma que se puede extraer directamente de él la aplicación lista para ser ejecutada. El presente trabajo ha

[8] https://www.microfocus.com

desarrollado un programa muy simple que lee este fichero XML intermedio y construye automáticamente toda la estructura de la aplicación en los respectivos IDEs.

Así, la aplicación automáticamente generada está lista para ser ejecutada en el correspondiente IDE: Eclipse para las aplicaciones JSF y Microsoft Visual Studio para las aplicaciones ASP.NET MVC. De esta forma, el código que implementa la capa de presentación y el código que implementa el control de acceso basado en roles (RBAC) se genera automáticamente, y gracias al principio de separación de problemas promovido por el patrón MVC, el código relacionado con las otras capas de la aplicación (si éste es modelado junto a los mapas navegacionales) puede ser fácilmente desarrollado en el respectivo IDE. Como se comentó, los perfiles WAE4x se centran en el aspecto navegacional de la capa de presentación Web, pero se puede usar UML estándar para modelar el resto de capas de una aplicación. Por tanto, las transformaciones automáticas generan el código de la capa de presentación y de las otras capas siempre que se utilicen clases UML estándar en los modelos.

Cabe destacar que en el trabajo realizado, la transformación automática de los modelos a código depende del motor de transformación específico de Borland Together, pero dado que los modelos son PSMs y pueden ser mantenidos con herramientas UML CASE genéricas, estos pueden evolucionar al igual que el código sin vincular su mantenimiento a ninguna herramienta concreta. Además, dado que las reglas de trasformación están bien definidas, estas dan control total sobre el código generado, permitiendo implementarlas en cualquier motor de transformación y/o ajustarlas de acuerdo a requisitos específicos que pueden ser incorporados durante el proceso de generación. Para demostrar esta capacidad, en las reglas de transformación especificadas en este trabajo se ha incorporado el patrón *Post-Redirect-Get* (PRG) (Chisholm, 2008), el cual impide los envíos duplicados de formularios y mejora la usabilidad por medio de la navegación a través del historial de los navegadores Web.

En resumen, los perfiles WAE4x permiten: (i) caracterizar la capa de presentación Web de aplicaciones empresariales desarrolladas con JSF o ASP.NET MVC usando perfiles UML; (ii) mejorar el desarrollo de código y la mantenibilidad a través de la inclusión explicita de patrones arquitectónicos y de diseño en los modelos UML; (iii) incluir el control de acceso basado en roles (RBAC) en los modelos; y (iv) soportar la implementación de reglas de transformación automáticas que permitan la generación de código desde los modelos, danto total control sobre su generación.

Estas propiedades son importantes porque, como se ha argumentado antes: (i) los frameworks JSF y ASP.NET MVC son dos de los más notables en el desarrollo de aplicaciones empresariales modernas; (ii) UML es la notación de diseño más importante promovida en todas las recomendaciones curriculares, el SWEBOK y los principales catálogos de patrones arquitectónicos y de diseño; (iii) la arquitectura multicapa es la más ampliamente usada en la industria y su implementación beneficia el proceso de desarrollo y el mantenimiento de las aplicaciones empresariales; (iv) las características RBAC deben ser integradas en la etapa de diseño de las aplicaciones de manera independiente de la lógica de negocio; (v) la generación automática de código es un objetivo importante en la ingeniería de software; y (vi) el enfoque presentado puede ser soportado por herramientas UML CASE genéricas, las más ampliamente usadas en la industria.

4.2 Desarrollo de los perfiles WAE4JSF y WAE4.NET

Como se ha comentado, UML-WAE es muy valioso para modelar capas de presentación Web. Sin embargo, aunque los modelos UML-WAE son diseños detallados que incluyen conceptos de programación, no proporcionan soporte para los componentes implementados en frameworks específicos tales como JSF y ASP.NET MVC, los cuales, a su vez, son ampliamente utilizados en el desarrollo de aplicaciones empresariales modernas.

Para soportar estos frameworks, el presente trabajo define los perfiles WAE4JSF y WAE4.NET. Estos perfiles extienden la definición de UML-WAE caracterizando los principales componentes relacionados con el aspecto navegacional de los frameworks JSF y ASP.NET MVC. De esta forma, los perfiles permiten modelar mapas navegacionales utilizando componentes específicos cuya semántica se define a través de la definición de reglas de transformación a los componentes implementados en los frameworks. Esta semántica facilita leer los modelos pensando en términos de los correspondientes componentes y además evita la ambigüedad en el modelado. Por tanto, las reglas de transformación definen una especificación ejecutable de los modelos clarificando su transformación a código y en última instancia cerrando la brecha entre modelos y código.

El proceso seguido para desarrollar los perfiles WAE4JSF y WAE4.NET, que se detalla en las siguientes secciones, puede ser reutilizado para soportar el desarrollo de otros aspectos y/o para otros frameworks de desarrollo, tales como Spring MVC. Al igual que en el proceso de desarrollo del perfil NMMp discutido en el Capítulo 3, este proceso se basa en las guías y recomendaciones prácticas dadas en (Fuentes & Vallecillo, 2004), las

cuales se han adaptado al desarrollo de los perfiles WAE4JSF y WAE4.NET y se resumen a continuación:

1. Definición del meta-modelo que representa los aspectos a modelar utilizando los mecanismos del propio UML o MOF (clases, relaciones de herencia, asociaciones, etc.). Este paso comprende la selección del framework de desarrollo y sus componentes que implementan los aspectos a modelar. El presente trabajo ha seleccionado el aspecto navegacional, la descripción de la estructura de la interfaz de usuario a través de plantillas, las cuales definen un diseño y comportamiento común, y el control de acceso basado en roles (RBAC) en dos de los frameworks más utilizados en el desarrollo de aplicaciones empresariales modernas: JSF y ASP.NET MVC. El meta-modelo define formalmente los componentes que implementan estos aspectos y sus relaciones.
2. Traducción del meta-modelo en un perfil UML. Este paso permite aprovechar todas los beneficios de UML en el proceso de desarrollo, principalmente, la habilidad de modelar los aspectos específicos en herramientas UML CASE genéricas. Además, este paso comprende la definición de la semántica de traducción entre los conceptos del meta-modelo y los estereotipos y valores etiquetados que los representa en un perfil UML.
3. Desarrollo de las reglas de transformación para la generación de código. Este paso comprende el desarrollo de reglas de transformación bien definidas para transformar los modelos UML en código conforme al framework seleccionado, habilitando de esta forma su directa implementación en motores de transformación específicos.

4.3 Perfil WAE4JSF

4.3.1 Meta-modelo MOF para WAE4JSF

El meta-modelo MOF mostrado en la Figura 4.1 (Cortés & Navarro, 2016) proporciona una especificación precisa para los elementos definidos en el perfil WAE4JSF. El meta-modelo caracteriza los componentes del framework JSF relacionados con el aspecto navegacional, la definición de la estructura de la interfaz de usuario a través de plantillas, y los componentes de control de acceso basado en roles (RBAC) soportados por el framework JAAS (Oracle, 2007). En la figura, los elementos definidos en UML-WAE han sido etiquetados con un estereotipo. Nótese que excepto los elementos *Forward* y *Redirect*, los restantes elementos de UML-WAE no pueden ser instanciados en los

modelos WAE4JSF. Esta decisión se debe a que el uso combinado de componentes JSF y componentes no JSF no es aconsejable en el desarrollo de aplicaciones JSF (Geary & Horstmann, 2010).

Desde el punto de vista navegacional, el presente trabajo define, *navegación estática* cuando la siguiente página en el proceso navegacional no depende de procesos computacionales (más allá de los llevados a cabo por el navegador y el servidor web), y *navegación dinámica* cuando la próxima página depende de la ejecución de lógica de negocio. La navegación estática se representa con los elementos *Static Link* y *Static Submit*, los cuales contienen un elemento *Outcome* como parte de su estructura. La navegación dinámica se representa con los elementos *Dynamic Link* y *Dynamic Submit*, los cuales tienen la capacidad de invocar un elemento *Action* definido como una operación de un elemento *Managed Bean*. Los elementos *Action* pueden retornar elementos *Outcome* de acuerdo a condiciones booleanas generadas por la ejecución de lógica de aplicación (posiblemente involucrando la capa de negocio) y representadas por elementos *Return Condition*. Tanto los elementos *Link* como los elementos *Submit* definen el atributo *Parameters*, el cual representa los datos que se transfieren entre los componentes de la capa de presentación.

La conexión entre la capa de presentación y la capa de negocio se representa por la dependencia desde el elemento *Managed Bean* al elemento *Business Logic*. Estos elementos son caracterizados por interfaces de software que pueden ser implementados localmente a través de servicios de aplicación (siguiendo el patrón *Application Services*) o por delegados de negocio (siguiendo el patrón *Business Delegate*) que representan servicios remotos. Debido a que WEA4JSF se restringe a la capa de presentación, el modelado de estos servicios, como los componentes de la capa de integración caen fuera del ámbito del perfil, pero pueden ser modelados usando UML estándar como se muestra en (Navarro et al., 2012).

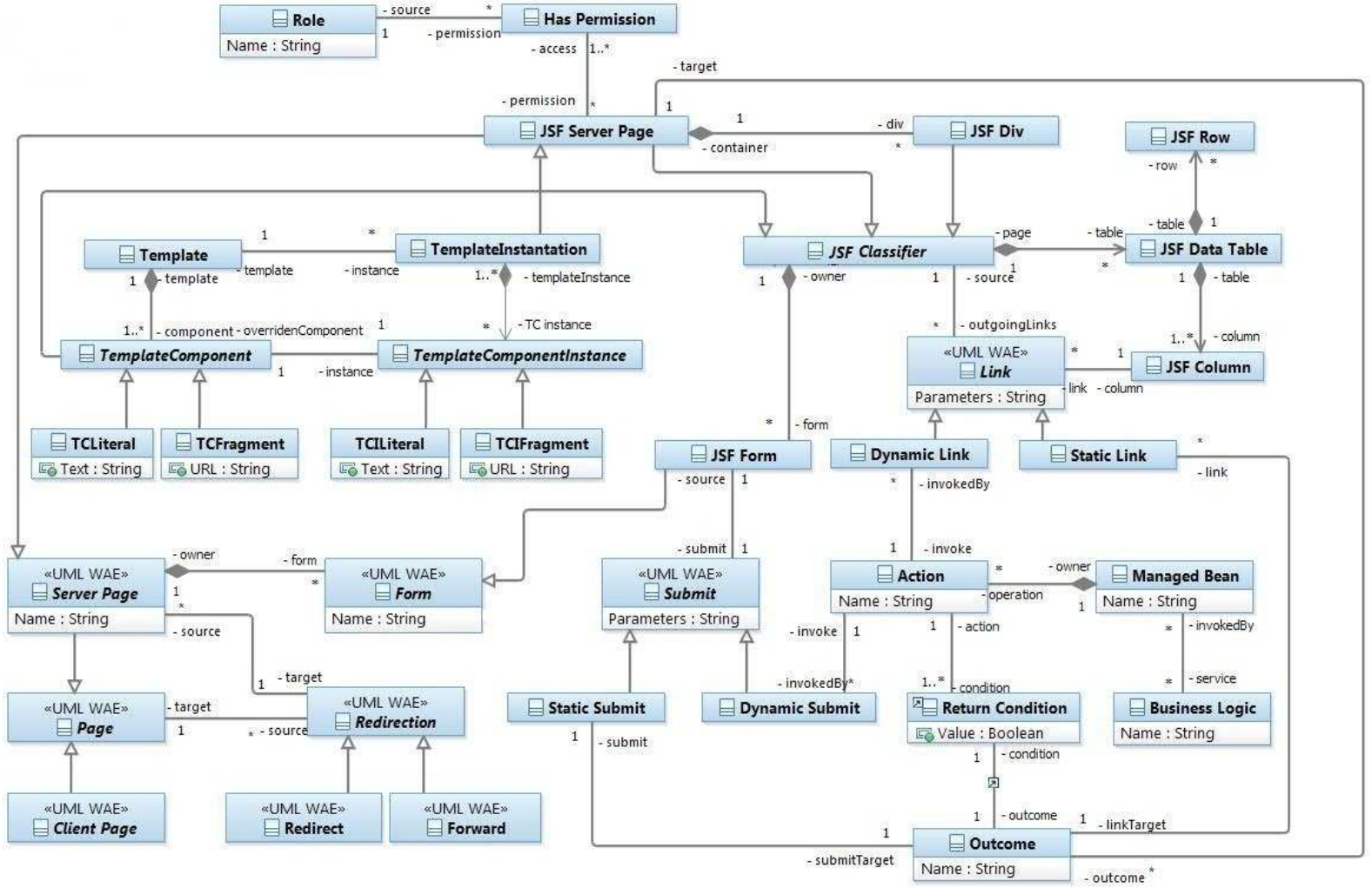
La asociación entre el elemento *Outcome* y el elemento *JSF Server Page* representa los componentes JSF responsables de usar elementos *Outcome* para encontrar la próxima página en el proceso navegacional, independientemente de si estos están definidos en los elementos estáticos o son producidos por los elementos dinámicos.

La descripción de la estructura de la interfaz de usuario a través de plantillas, las cuales definen un diseño y comportamiento común, se modela usando los elementos *Template* y *Template Component*, los cuales caracterizan las plantillas y sus regiones

respectivamente, definiendo además el contenido común que puede ser literal (*TCLiteral*) o cargado usando una URL (*TCFragment*). Los elementos *JSF Server Page* pueden instanciar una plantilla a través del elemento *TemplateInstantation*, reutilizando su contenido común y pudiendo sobrescribir las regiones que éste define a través del elemento *TemplateComponenentInstance* para agregar su propio contenido, el cual, también puede ser literal (*TCILiteral*) o cargado usando una URL (*TCIFragment*). Nótese que el elemento *TemplateInstantation* hereda del elemento *JSF Server Page* y por tanto comparte su estructura y comportamiento.

Finalmente, el control de acceso basado en roles (RBAC) se representa con el elemento *Has Permission* que relaciona a los elementos *Role* y *JSF Server Page* especificando qué roles en el dominio de la aplicación tienen permisos sobre qué páginas. La siguiente sección muestra la representación del meta-modelo de la Figura 4.1 en términos de estereotipos y valores etiquetados del perfil WAE4JSF y algunos ejemplos de su uso.

Figura 4.1: Meta-modelo MOF para WAE4JSF



4.3.2 Perfil UML para WAE4JSF

Las Tablas 4.1 a 4.26 describen en detalle los conceptos especificados en el meta-modelo de la Figura 4.1. Nótese que para la mayoría de ellos se define un estereotipo, con los correspondientes valores etiquetados e icono para su representación visual en los diagramas, que al final constituyen el perfil WAE4JSF.

Tabla 4.1: Concepto JSF Server Page Fragment.

Concepto WAE4JSF	**JSF Classifier**
Estereotipo	-
Elemento UML	-
Descripción	Clasificador abstracto que define la estructura y comportamiento común de los elementos *JSF Server Page*, *JSF Div*, *TemplateComponent* y *TemplateInstantiation.*
Valores etiquetados	-

Tabla 4.2: Estereotipo WAE4JSF JSF Server Page.

Concepto WAE4JSF	**JSF Server Page**
Estereotipo	JSF Server Page
Elemento UML	Clase
Icono	
Descripción	Páginas JSF. Este elemento caracteriza las páginas procesadas por el servidor antes de ser enviadas al cliente. JSF no considera páginas directamente enviadas sin procesamiento.
Valores etiquetados	-

Tabla 4.3: Estereotipo WAE4JSF JSF Form.

Concepto WAE4JSF	**JSF Form**
Estereotipo	JSF Form
Elemento UML	Clase

Icono	
Descripción	Formulario JSF. Debido a que el envío de formularios por el método GET no está soportado en JSF, todos los envíos de formulario se implementan con el método POST.
Valores etiquetados	-

Tabla 4.4: Estereotipo WAE4JSF Outcome.

Concepto WAE4JSF	**Outcome**
Estereotipo	Outcome
Elemento UML	Clase
Icono	
Descripción	JSF outcome. Elemento que da acceso a páginas destino en el proceso navegacional, de acuerdo a reglas preestablecidas. Estos elementos pueden definirse estáticamente o ser generados dinámicamente.
Valores etiquetados	-

Tabla 4.5: Estereotipo WAE4JSF Access.

Concepto WAE4JSF	**Asociación entre Outcome y JSF Server Page**
Estereotipo	Access
Elemento UML	Asociación dirigida
Descripción	Indica el acceso a una página JSF por medio de un outcome independientemete si el outcome se define estática o dinámicamente.
Valores etiquetados	-

Tabla 4.6: Estereotipo WAE4JSF Managed Bean.

Concepto WAE4JSF	**Managed Bean**
Estereotipo	Managed Bean

Elemento UML	Clase
Icono	
Descripción	JSF managed bean. Clase de presentación Java que puede ser invocada por páginas JSF para ejecutar lógica de presentación.
Valores etiquetados	-

Tabla 4.7: Concepto WAE4JSF Action.

Concepto WAE4JSF	**Action**
Estereotipo	-
Elemento UML	-
Descripción	Acción definida en un managed bean para ejecutar lógica de presentación. Este elemento puede ser invocado por los elementos que implementan navegación dinámica (*Dynamic Link* y *Dynamic Submit*).
Valores etiquetados	-

Tabla 4.8: Estereotipo WAE4JSF Static Link.

Concepto WAE4JSF	**Static Link**
Estereotipo	Static Link
Elemento UML	Asociación dirigida
Descripción	Relación navegacional estática desde un *JSF Classifier* a una página JSF a través de un outcome. Este tipo de relación navegacional no requiere la ejecución de lógica de aplicación.
Valores etiquetados	*Parameters*. Usado para mover datos entre las paginas.

Tabla 4.9: Estereotipo WAE4JSF Dynamic Link.

Concepto WAE4JSF	**Dynamic Link**
Estereotipo	Dynamic Link
Elemento UML	Asociación dirigida

Descripción	Relación navegacional dinámica desde un *JSF Classifier* a una página JSF a través de una acción definida en un managed bean. La acción retorna outcomes de acuerdo a condiciones booleanas alcanzadas por la ejecución de lógica de aplicación. Este tipo de relación navegacional requiere la ejecución de lógica de aplicación.
Valores etiquetados	*Action* y *Parameters*. El primero describe el nombre de la acción a invocar en el managed bean. El segundo se utiliza para mover datos entre la página y el managed bean.

Tabla 4.10: Estereotipo WAE4JSF Dynamic Submit.

Concepto WAE4JSF	**Dynamic Submit**
Estereotipo	Dynamic Submit
Elemento UML	Asociación dirigida
Descripción	Caracteriza formularios JSF que envían datos a un managed bean a través del método POST generando navegación dinámica, la cual requiere de ejecución de lógica de aplicación.
Valores etiquetados	*Action* y *Parameters*. El primero describe el nombre de la acción a invocar en el managed bean. El segundo se utiliza para mover los datos definidos en el formulario al managed bean.

Tabla 4.11: Estereotipo WAE4JSF Return Condition.

Concepto WAE4JSF	**Return Condition**
Estereotipo	Return
Elemento UML	Asociación dirigida
Descripción	Un manged bean que retorna un outcome de acuerdo a una condición booleana alcanzada por la lógica de negocio.
Valores etiquetados	*Condition.* Describe la condición booleana que debe ser alcanzada para retornar el outcome.

Tabla 4.12: Estereotipo WAE4JSF Template.

Concepto WAE4JSF	**Template**
Estereotipo	Template

Elemento UML	Clase
Icono	
Descripción	Página plantilla o base que define el diseño y comportamiento común reutilizable por otras páginas en una aplicación.
Valores etiquetados	-

Tabla 4.13: Estereotipo WAE4JSF TCLiteral.

Concepto WAE4JSF	**TCLiteral**
Estereotipo	TCLiteral
Elemento UML	Clase
Icono	
Descripción	Región definida dentro de una plantilla. El contenido que define su estructura y comportamiento es completamente definido dentro del elemento.
Valores etiquetados	*Text*. Usado para describir su contenido.

Tabla 4.14: Estereotipo WAE4JSF TCFragment.

Concepto WAE4JSF	**TCFragment**
Estereotipo	TCFragment
Elemento UML	Clase
Icono	
Descripción	Región definida dentro de una plantilla. Su contenido es cargado desde elementos externos.
Valores etiquetados	*URL*. Usado para describir el recurso desde donde se carga su contenido.

Tabla 4.15: Estereotipo WAE4JSF TemplateInstantation.

Concepto WAE4JSF	**TemplateInstantation**

Estereotipo	TemplateInstantation
Elemento UML	Clase
Icono	
Descripción	Componente JSF capaz de instanciar una plantilla permitiendo reutilizar sus regiones y sobrescribir algunas de ellas.
Valores etiquetados	-

Tabla 4.16: Estereotipo WAE4JSF TCILiteral.

Concepto WAE4JSF	**TCILiteral**
Estereotipo	TCILiteral
Elemento UML	Clase
Icono	
Descripción	Región que, definida en una página JSF que instancia una plantilla, sobrescribe a una región definida en la plantilla. El contenido que define su estructura y comportamiento es completamente definido dentro del elemento.
Valores etiquetados	*Text*. Usado para describir su contenido.

Tabla 4.17: Estereotipo WAE4JSF TCIFragment.

Concepto WAE4JSF	**TCIFragment**
Estereotipo	TCIFragment
Elemento UML	Clase
Icono	
Descripción	Región que, definida en una página JSF que instancia una plantilla, sobrescribe a una región definida en la plantilla. Su contenido es cargado desde elementos externos.
Valores etiquetados	*URL*. Usado para describir el recurso desde donde se carga su contenido.

Tabla 4.18: Estereotipo WAE4JSF TemplateInstance.

Concepto WAE4JSF	**Asociación entre TemplateInstantiation y Template**
Estereotipo	TemplateInstance
Elemento UML	Dependency
Descripción	Una página JSF que instancia una plantilla para reutilizar el contenido y comportamiento definido en sus regiones.
Valores etiquetados	-

Tabla 4.19: Estereotipo WAE4JSF JSF Data Table.

Concepto WAE4JSF	**JSF Data Table**
Estereotipo	JSF Data Table
Elemento UML	Clase
Icono	
Descripción	Componente JSF para representar datos tabulares.
Valores etiquetados	-

Tabla 4.20: Estereotipo WAE4JSF JSF Div.

Concepto WAE4JSF	**JSF Div**
Estereotipo	JSF Div
Elemento UML	Clase
Icono	
Descripción	Componente JSF que representa secciones o divisiones definidas dentro de una página JSF y que comparten su estructura y comportamiento.
Valores etiquetados	-

Tabla 4.21: Estereotipo WAE4JSF JSF Row.

Concepto WAE4JSF	**JSF Row**
Estereotipo	JSF Row

Elemento UML	Clase
Descripción	Componente JSF que representa un ítem dentro de los datos tabulares.
Valores etiquetados	-

Tabla 4.22: Estereotipo WAE4JSF JSF Column.

Concepto WAE4JSF	**JSF Column**
Estereotipo	JSFColumn
Elemento UML	Clase
Icono	
Descripción	Componente JSF que representa una propiedad de un ítem dentro de los datos tabulares.
Valores etiquetados	-

Tabla 4.23: Estereotipo WAE4JSF Role.

Concepto WAE4JSF	**Role**
Estereotipo	Role
Elemento UML	Clase
Icono	
Descripción	Role JAAS definido dentro del dominio de la aplicación.
Valores etiquetados	-

Tabla 4.24: Concepto WAE4JSF Has Permission.

Concepto WAE4JSF	**Has Permission**
Estereotipo	HasPermission
Elemento UML	Asociación dirigida
Descripción	Una página JSF que puede ser accedida por un rol JAAS.
Valores etiquetados	-

Tabla 4.25: Estereotipo WAE4JSF Business Logic.

Concepto WAE4JSF	**Business Logic**
Estereotipo	BusinessLogic
Elemento UML	Clase
Descripción	Componente de lógica de negocio. Utilizado para describir el punto de entrada desde la capa de presentación a las capas internas de la aplicación.
Valores etiquetados	-

Tabla 4.26: Concepto UML-WAE Redirect.

Concepto WAE4JSF	**UML-WAE Redirect**
Estereotipo	Redirect
Elemento UML	Asociación dirigida
Descripción	Redirección HTTP
Valores etiquetados	-

Tabla 4.27: Concepto UML-WAE Forward

Concepto WAE4JSF	**UML-WAE Forward**
Estereotipo	Forward
Elemento UML	Asociación dirigida
Descripción	JSF Forward
Valores etiquetados	-

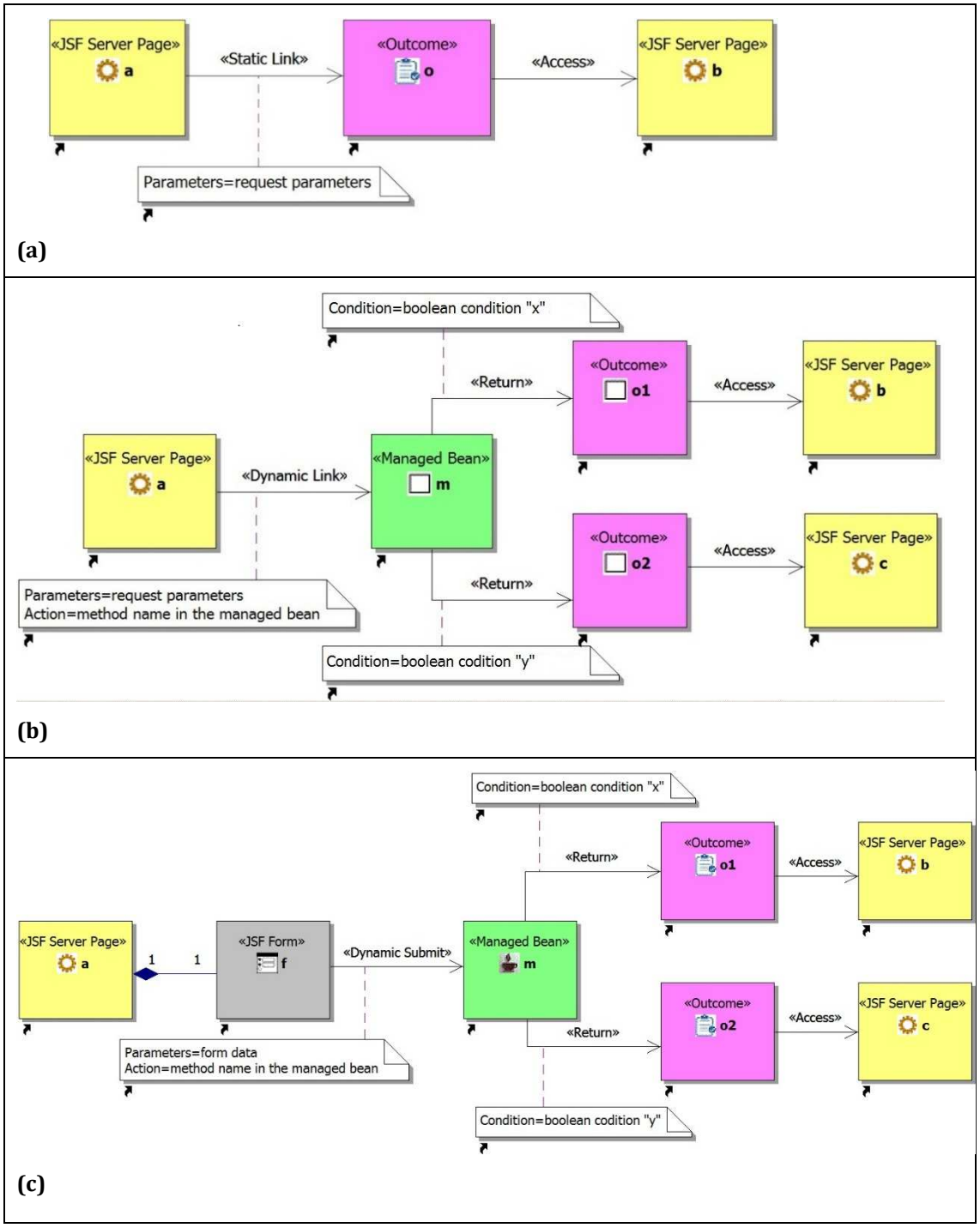
La Figura 4.2 ilustra algunos de estos conceptos. En la Figura 4.2a, una *JSF Server Page a* se relaciona con un *Outcome* por medio de una asociación estereotipada con *Static Link* para representar la navegación estática desde la página *a* hasta la página *b*, la cual es accedida por el *Outcome o* a través de la asociación estereotipada con *Access*. La navegación se ejecuta por medio de una petición de tipo GET que puede transportar datos en el valor etiquetado *Parameters* del estereotipo *Static Link.*

En la Figura 4.2b, una *JSF Server Page a* se relaciona con un *Managed Bean* a través de una asociación estereotipada con *Dynamic Link* para representar la navegación dinámica

desde la página *a* hasta una de las páginas accedidas por uno de los *Outcome* retornados por el *Managed Bean m*. La página destino depende de la condición booleana alcanzada en la ejecución del método especificado en el valor etiquetado *Action* del estereotipo *Dynamic Link*. De acuerdo a la condición booleana, el método retorna el *Outcome* que se usa para acceder a la página destino. La navegación se ejecuta por medio de una petición de tipo GET que puede transportar datos en el valor etiquetado *Parameters* del estereotipo *Dynamic Link*.

Finalmente, en la Figura 4.2c una *JSF Server Page a*, la cual contiene un *JSF Form f*, se relaciona con un *Managed Bean* a través de una asociación estereotipada con *Dynamic Submit* para representar la navegación dinámica desde la página *a* hasta una de las páginas accedidas por uno de los *Outcome* retornados por el *Managed Bean m*. La página destino depende de la condición booleana alcanzada en la ejecución del método especificado en el valor etiquetado *Action* del estereotipo *Dynamic Submit*. De acuerdo a la condición booleana, el método retorna el *Outcome* que se usa para acceder a la página destino. La navegación se ejecuta por medio de una petición de tipo POST que transportar los datos del formulario en el valor etiquetado *Parameters* del estereotipo *Dynamic Submit*.

Figura 4.2: Ejemplos de uso de WAE4JSF. (a) Navegación estática vía GET con WAE4JSF. (b) Navegación dinámica vía GET con WAE4JSF. (c) Navegación dinámica vía POST con WAE4JSF. Los valores etiquetados se muestran como notas UML



4.3.3 Transformación WAE4JSF a código

Siguiendo el enfoque del desarrollo dirigido por modelos (MDD), el presente trabajo desarrolla un conjunto de reglas de transformación modelo a texto (M2T) que permiten transformar los elementos de los modelos WAE4JSF en código de capa de presentación JSF. La Tabla 4.28 especifica estas reglas de transformación explícitamente, lo cual

posibilita la transformación manual, y hace que puedan ser fácilmente implementadas en cualquier motor de transformación. Para demostrar esta característica, el presente trabajo ha implementado estas reglas conforme al motor de transformaciones M2T implementado en Borland Together. El Anexo 8.1 muestra la implementación de algunas de estas reglas, las cuales, son específicas de Borland Together y se definen en términos de transformaciones XSLT que usan expresiones OCL para seleccionar elementos de los modelos y generar ficheros de texto a partir de ellos que constituyen el código de las aplicaciones JSF.

Por medio de la ejecución de las reglas de transformación, los modelos WAE4JSF se transforman en: (i) componentes JSF relacionados con la capa de presentación Web, incluyendo páginas JSF, managed beans y secciones en el fichero de configuración *faces-config.xml*; (ii) secciones de configuración JAAS en el fichero *web.config* para soportar el control de acceso basado en roles (RBAC) a nivel de páginas: (iii) clases Java que se corresponden con las clases UML estándar incluidas en los modelos; y (iv) la estructura de directorios siguiendo la disposición estandarizada por Java que permite que la aplicación JSF pueda ser directamente desplegada en Eclipse. Por supuesto, las páginas JSF deben ser complementadas con el contenido que define la estructura de presentación y las clases pertenecientes a las capas de negocio e integración deben ser implementadas con el código de la aplicación, pero el esquema de las páginas JSF, incluyendo formularios, botones de comandos, enlaces (*links*) y managed beans se generan por la aplicación (manual o automática) de estas reglas.

Tabla 4.28: Transformación WAE4JSF a componentes JSF.

Estereotipo WAE4JSF	**Componente JSF**	**Descripción**
JSF Server Page	JSF Page	La estructura de paquetes UML que contiene cada *JSF Server Page* se traduce a una estructura de directorios. Un elemento *JSF Server Page* se corresponde esencialmente con un fichero HTML con etiquetas JSF
JSF Form	Componente JSF h:form	Formulario JSF que contiene datos presentados en la interfaz de usuario y tiene la capacidad de enviarlos al servidor Web

Estereotipo WAE4JSF	**Componente JSF**	**Descripción**
Outcome	Atributo outcome definido en los componentes JSF capaces de lanzar un proceso navegacional	En la navegación estática, los outcome son directamente enlazados en los componentes JSF capaces de lanzar el proceso navegacional
Asociación *Access* entre *Outcome* y *JSF Server Page*	Elemento *navigation-case* en una regla de navegación especificada dentro del fichero *faces-config.xml*	El elemento *navigation-case* es un elemento padre de los elementos *from-outcome* y *to-view-id*. El *Outcome* fuente de la asociación se usa para construir el elemento *from-outcome*, y la *JSF Server Page* destino de la relación se usa para construir el elemento *to-view-id*
Managed Bean	Clase managed bean y elemento *managed-bean* especificado en el fichero *faces-config.xml*, el cual define el nombre de la clase completamente calificado (espacio de nombre y ámbito)	La estructura de paquetes que contiene la clase que define el managed bean se traduce a una estructura de directorios. El elemento *managed-bean* generado permite a JSF localizar e instanciar los managed beans
Asociación dirigida *Return* entre un *Managed Bean* y un *Outcome*	Descripción de un método público en el managed bean	El método público en el managed bean fuente de la asociación debe retornar una cadena que representa al *Outcome* del destino de la asociación, el cual se usa para determinar la próxima pagina
Asociación dirigida *Static Link* entre una *JSF Server Page* y un *Outcome*	Componente JSF h:link dentro de la página JSF implementando la navegación estática ejecutada por medio del método GET	El componente h:link contiene el atributo outcome. El *Outcome* destino de la asociación se usa para construir el contenido de este atributo implementado la navegación estática. El valor etiquetado *Parameters* definido en el estereotipo *Static Link* se usa para especificar los datos enviados entre las paginas JSF

Estereotipo WAE4JSF	**Componente JSF**	**Descripción**
Asociación dirigida *Dinamic Link* entre una *JSF Server Page* y un *Managed Bean*	Componente JSF h:link dentro de la página JSF implementando la navegación dinámica ejecutada por medio del método GET	El componente h:link define el atributo action. El *Managed Bean* destino de la asociación y el valor etiquetado *Action* definido en el estereotipo *Dinamic Link* se usan para construir la expresión definida en el atributo outcome implementando la navegación dinámica. El valor etiquetado *Parameters* definido en el estereotipo *Dinamic Link* se usa para especificar los datos enviados entre la página y el managed bean
Asociación dirigida *Dynamic Submit* entre un *JSF Form* y un *Managed Bean*	Componente JSF h:comandButton dentro de formulario JSF implementando la navegación dinámica por medio del método POST	El componente JSF h:comandButton define el atributo action. El *Managed Bean* destino de la asociación y el valor etiquetado *Action* definido en el estereotipo *Dynamic Submit* se usan para construir la expresión definida en el atributo *action* implementado la navegación dinámica. El valor etiquetado *Parameters* definido en el estereotipo *Dynamic Submit* se usa para especificar los datos enviados desde el formulario al *Managed Bean*
Composición entre *Template* y *TCLiteral/TCFragment*	Plantilla JSF capaz de especificar regiones con contenido por defecto a través del elemento ui:include. Cada *TCLiteral/TCFragment* representa una región con contenido por defecto	El elemento ui:include define las regiones en la plantilla JSF. Este elemento define el atributo src cuyo contenido se crea a partir de del valor etiquetado *Text* o *URL* de los componentes *TCLiteral/TCFragment* respectivamente.
Asociación dirigida *Template Instance* entre un *Template*	Página JSF configurada para cargar una plantilla a través del elemento ui:composition e incluir	El elemento ui:composition permite cargar una plantilla en una página JSF. este elemento define el atributo template

Estereotipo WAE4JSF	Componente JSF	Descripción
Instantation y un *Template*	contenido que sobrescriba algunas regiones por defecto	cuyo contenido se crea a partir del *Template* destino de la asociación
Dependencia entre un *TCILiteral/TCIFragment* a un *TCLiteral/TCFragment*	Elemento ui:define en la página JSF que contiene el *TCILiteral/TCIFragment*	El elemento ui:define permite sobrescribir una región definida en una plantilla para agregar contenido propio. Este elemento define el atributo name que especifica el nombre de la región a sobrescribir y cuyo contenido se crea a partir del *TCLiteral/TCFragment* destino de la dependencia. El contenido del elemento ui:define se crea a partir del *TCILiteral/TCIFragment* fuente de la dependencia
Role	Elemento JAAS security-role definido en el fichero web-config.xml	El elemento JAAS security-role contiene el elemento role-name. La clase estereotipada con *Role* se usa para construir el contenido de este elemento
Asociación dirigida *Has Permision* entre un *Role* y una *JSF Server Page*	Elemento JAAS security-constraint en el fichero web-config.xml	El elemento JAAS security-constraint contiene el elemento web-resource-collection. El *JSF server page* destino de la asociación es usado para construir el contenido de este elemento. Además el elemento JAAS security-constraint contiene el elemento auth-constraint. El *Role* fuente de la relación es usado para construir el contenido de este elemento

4.4 Perfil WAE4.NET

4.4.1 Meta-modelo MOF para WAE4.NET

La Figura 4.3 muestra el meta-modelo MOF (Cortés & Navarro, 2016) definido en este trabajo para caracterizar los componentes del framework ASP.NET MVC relacionados con el aspecto navegacional, la definición de la estructura de la interfaz de usuario a través de plantillas y los componentes de control de acceso basado en roles (RBAC) soportados

por el framework. En ASP.NET MVC, la navegación se ejecuta por medio de peticiones desde las páginas Web a métodos, llamados acciones (*actions*), definidos en clases controlador (*controllers*). Las acciones implementan la lógica de la aplicación (involucrando posiblemente a la capa de negocio) y encuentran y retornan la próxima página en el proceso navegacional. Por tanto, ASP.NET MVC solo soporta la navegación dinámica (Esposito, 2011).

En la Figura 4.3, los elementos de UML-WAE han sido etiquetados con un estereotipo. Nótese que ninguno de los elementos de UML-WAE pueden ser instanciados directamente en los modelos WAE4.NET, debido a que en ASP.NET MVC no se permite el uso de componentes no definidos por el framework.

La relación entre una página fuente y una acción se representa con el elemento *Action Link*, y la relación entre un formulario contenido en una página y una acción se representa con el elemento *Action Submit*. Tanto el elemento *Link* como *Submit* definen el atributo *Paramenters*, el cual representa los objetos de transferencia de datos que transmiten los datos entre las páginas y las acciones, y que en el contexto de ASP.NET MVC se conocen como *Input Model*.

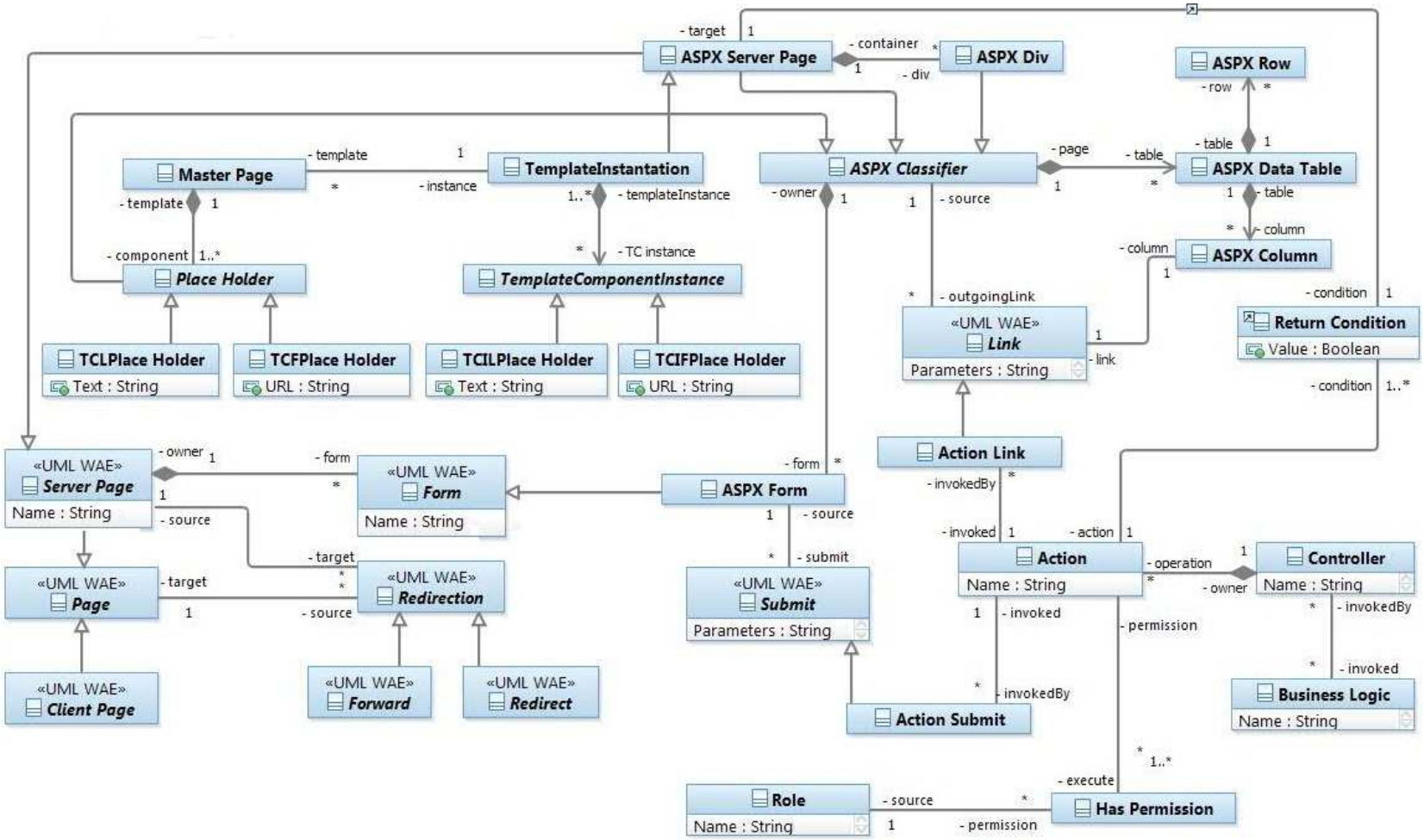
La asociación entre elementos *Action* y elementos *ASPX Server Page* representan los componentes ASP.NET MVC responsables de encontrar la siguiente página en el proceso navegacional de acuerdo a condiciones booleanas representadas por elementos *Return Condition*. Esta asociación también define los datos que se usan para crear el contenido de la página destino, y que en el contexto de ASP.NET MVC se conocen como *View Models*. Las condiciones booleanas se obtienen por la ejecución de la lógica de aplicación caracterizada por el elemento *Business Logic*, el cual tiene la misma semántica que en WAE4JSF.

La descripción de la estructura de la interfaz de usuario a través de plantillas, las cuales definen un diseño y comportamiento común, se modela usando los elementos *Master Page* y *Place Holder*, los cuales caracterizan las plantillas y sus regiones respectivamente, definiendo el contenido común que puede ser literal (*TCLPlace Holder*) o cargado usando una URL (*TCFPlace Holder*). Los elementos *ASPX Server Page* pueden instanciar una plantilla a través del elemento *TemplateInstantation*, reutilizando su contenido común y pudiendo sobrescribir las regiones que éste define a través del elemento *TemplateComponenetInstance* para agregar su propio contenido, el cual, también puede ser literal (*TCILPlace Holder*) o ser cargado usando una URL (*TCIFPlace Holder*).

Nótese que el elemento *TemplateInstantation* hereda del elemento *ASPX Server Page* y por tanto comparte su estructura y comportamiento.

Finalmente, el control de acceso basado en roles (RBAC) se representa con el elemento *Has Permission,* que relaciona a los elementos *Role* y *Action,* especificando qué roles en el dominio de la aplicación tienen permisos de ejecución sobre qué acciones. La siguiente sección muestra la representación del meta-modelo de la Figura 4.3 en términos de estereotipos y valores etiquetados del perfil WAE4.NET y algunos ejemplos de su uso.

Figura 4.3: Meta-modelo MOF para WAE4.NET



4.4.2 Perfil UML para WAE4.NET

Las Tablas 4.29 a 4.50 describen en detalle los conceptos especificados en el meta-modelo de la Figura 4.3. Nótese que para la mayoría de ellos se define el estereotipo, con los correspondientes valores etiquetados e icono para su representación visual en los diagramas, que al final constituyen el perfil WAE4.NET.

Tabla 4.29: Concepto ASPX Server Page Fragment.

Concepto WAE4.NET	**ASPX Classifier**
Estereotipo	-
Elemento UML	-
Descripción	Clasificador abstracto que define la estructura y comportamiento común de los elementos *ASPX Server Page*, *ASPX Div*, *Place Holder* y *TemplateInstantiation*.
Valores etiquetados	-

Tabla 4.30: Estereotipo WAE4.NET ASPX Server Page.

Concepto WAE4.NET	**ASPX Server Page**
Estereotipo	ASPX Server Page
Elemento UML	Clase
Icono	
Descripción	Página ASPX. Este elemento caracteriza las páginas procesadas por el servidor antes de ser enviadas al cliente. ASP.NET MVC no considera páginas directamente enviadas sin procesamiento.
Valores etiquetados	-

Tabla 4.31: Estereotipo WAE4.NET ASPX Form.

Concepto WAE4.NET	**ASPX Form**
Estereotipo	ASPX Form
Elemento UML	Clase

Icono	
Descripción	Formulario ASPX. Debido a que el envío de formularios por el método GET no está soportado en ASP.NET MVC, todos los envíos de formulario se implementan con el método POST.
Valores etiquetados	-

Tabla 4.32: Estereotipo WAE4.NET Controller.

Concepto WAE4.NET	**Controller**
Estereotipo	Controller
Elemento UML	Clase
Icono	
Descripción	Controlador ASP.NET MVC. Clase de capa de presentación .NET que puede ser invocada por páginas ASPX para ejecutar lógica de presentación.
Valores etiquetados	-

Tabla 4.33: Estereotipo WAE4.NET Action.

Concepto WAE4.NET	**Action**
Estereotipo	Action
Elemento UML	Clase
Icono	
Descripción	Método ASP.NET MVC definido en una clase controlador, conocido como acción. Las acciones en .NET son responsables de recibir las solicitudes desde las páginas ASPX, ejecutar lógica de aplicación y retornar la siguiente página en el proceso navegacional.
Valores etiquetados	-

Tabla 4.34: Estereotipo WAE4.NET Action Link.

Concepto WAE4.NET	**Action Link**
Estereotipo	Action Link
Elemento UML	Asociación dirigida
Descripción	Relación navegacional dinámica definida entre un *ASPX Classifier* y una página ASPX por medio de una acción. Este tipo de navegación requiere de ejecución de lógica de aplicación.
Valores etiquetados	-

Tabla 4.35: Estereotipo WAE4.NET Action Submit.

Concepto WAE4.NET	**Action Submit**
Estereotipo	Action Submit
Elemento UML	Asociación dirigida
Descripción	Un formulario ASPX que envía sus datos a una acción, generando navegación dinámica, la cual requiere ejecución de lógica de aplicación.
Valores etiquetados	-

Tabla 4.36: Estereotipo WAE4.NET Return.

Concepto WAE4.NET	**Return Condition**
Estereotipo	Return
Elemento UML	Asociación dirigida
Descripción	Navegación a una página ASPX desde una acción de acuerdo a una condición booleana alcanzada por medio de la ejecución de lógica de la aplicación.
Valores etiquetados	*Condition.* Describe la condición boolena que debe ser alcanzada para retornar la página ASPX.

Tabla 4.37: Estereotipo WAE4.NET Master Page.

Concepto WAE4.NET	**Master Page**
Estereotipo	Master Page

Elemento UML	Clase
Icono	
Descripción	Página plantilla o base que define el diseño y comportamiento común reutilizable por otras páginas en una aplicación.
Valores etiquetados	-

Tabla 4.38: Estereotipo WAE4.NET TCLPlace Holder.

Concepto WAE4.NET	**TCLPlace Holder**
Estereotipo	TCLPlace Holder
Elemento UML	Clase
Icono	
Descripción	Región definida dentro de una plantilla. El contenido que define su estructura y comportamiento se define complemente dentro del elemento.
Valores etiquetados	*Text.* Usado para describir su contenido.

Tabla 4.39: Estereotipo WAE4.NET TCFPlace Holder.

Concepto WAE4.NET	**TCFPlace Holder**
Estereotipo	TCFPlace Holder
Elemento UML	Clase
Icono	
Descripción	Región definida dentro de una plantilla. Su contenido se carga desde elementos externos.
Valores etiquetados	*URL*. Usado para describir el recurso desde donde se carga su contenido.

Tabla 4.40: Estereotipo WAE4.NET Template Instantation.

Concepto WAE4.NET	**TemplateInstantation**
Estereotipo	TemplateInstantation
Elemento UML	Clase
Icono	
Descripción	Componente ASP.NET MVC capaz de instanciar una plantilla permitiendo reutilizar sus regiones y sobrescribir algunas de ellas.
Valores etiquetados	-

Tabla 4.41: Estereotipo WAE4.NET TCILPlace Holder.

Concepto WAE4.NET	**TCILPlace Holder**
Estereotipo	TCILPlace Holder
Elemento UML	Clase
Icono	
Descripción	Región que, definida en una página ASPX que instancia *Master Page*, sobrescribe a una región definida en la plantilla. El contenido que define su estructura y comportamiento se define completamente dentro del elemento.
Valores etiquetados	*Text*. Usado para describir su contenido.

Tabla 4.42: Estereotipo WAE4.NET TCIFPlace Holder.

Concepto WAE4.NET	**TCIFPlace Holder**
Estereotipo	TCIFPlace Holder
Elemento UML	Clase
Icono	
Descripción	Región que, definida en una página ASPX que instancia una *Master Page*, sobrescribe a una región definida en la plantilla. Su contenido se carga desde elementos externos.

Valores etiquetados	*URL*. Usado para describir el recurso desde donde se carga su contenido.

Tabla 4.43: Estereotipo WAE4.NET Template Instance.

Concepto WAE4.NET	**Asociación entre TemplateInstantation y Master Page**
Estereotipo	TemplateInstance
Elemento UML	Dependency
Descripción	Una página ASPX que instancia una *Master Page* para reutilizar el contenido y comportamiento definido en sus regiones.
Valores etiquetados	-

Tabla 4.44: Estereotipo WAE4.NET ASPX Data Table.

Concepto WAE4.NET	**ASPX Data Table**
Estereotipo	ASPX Data Table
Elemento UML	Clase
Icono	
Descripción	Componente ASP.NET MVC para representar datos tabulares.
Valores etiquetados	-

Tabla 4.45: Estereotipo WAE4.NET ASPX Div.

Concepto WAE4.NET	**ASPX Div**
Estereotipo	ASPX Div
Elemento UML	Clase
Icono	
Descripción	Componente ASP.NET MVC que representa secciones o divisiones definidas dentro de una página ASPX y que comparten su estructura y comportamiento.
Valores etiquetados	-

Tabla 4.46: Estereotipo WAE4.NET ASPX Row.

Concepto WAE4.NET	**ASPX Row**
Estereotipo	ASPX Row
Elemento UML	Clase
Descripción	Componente ASP.NET MVC que representa un ítem dentro de los datos tabulares.
Valores etiquetados	-

Tabla 4.47: Estereotipo WAE4.NET ASPX Column.

Concepto WAE4.NET	**ASPX Column**
Estereotipo	ASPX Column
Elemento UML	Clase
Icono	
Descripción	Componente ASP.NET MVC que representa una propiedad de un ítem dentro de los datos tabulares.
Valores etiquetados	-

Tabla 4.48: Estereotipo WAE4.NET Role.

Concepto WAE4.NET	**Role**
Estereotipo	Role
Elemento UML	Clase
Icono	
Descripción	Rol ASP.NET MVC definido dentro del dominio de la aplicación.
Valores etiquetados	-

Tabla 4.49: Estereotipo WAE4.NET Has Permission.

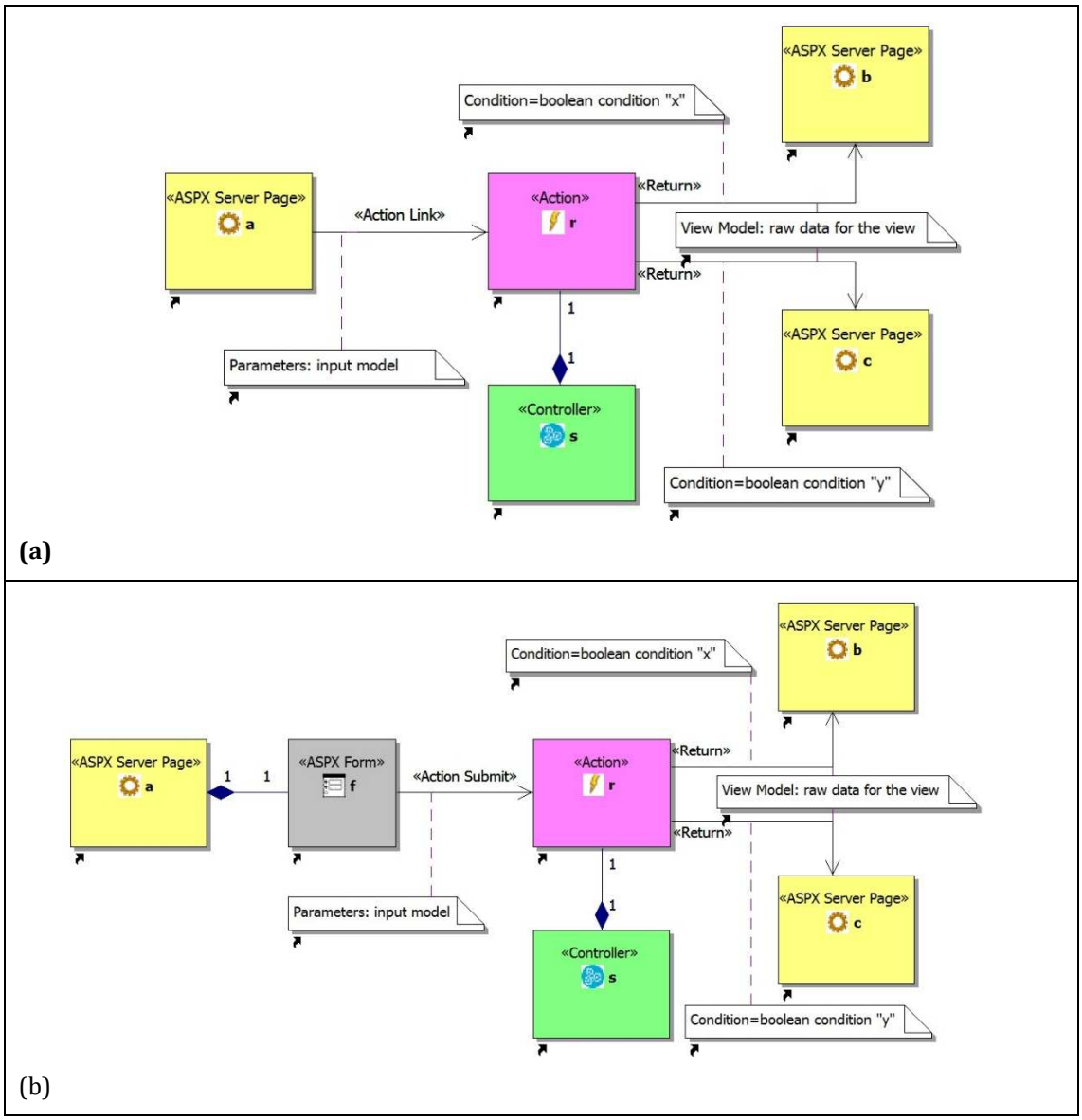
Concepto WAE4.NET	**Has Permission**
Estereotipo	Has Permission

Elemento UML	Asociación dirigida
Descripción	Una acción que puede ser ejecutada por un rol. Esta relación caracteriza las páginas ASPX que pueden ser accedidas por un Rol.
Valores etiquetados	-

Tabla 4.50: Estereotipo WAE4.NET Business Logic.

Concepto WAE4.NET	**Business Logic**
Estereotipo	Business Service
Elemento UML	Clase
Descripción	Componente de lógica de negocio.
Valores etiquetados	-

Figura 4.4: Ejemplos de uso de WAE4.NET. (a) Action Link con el perfil WAE4.NET. (b) Action Submit con el perfil WAE4.NET. Los valores etiquetados se muestran como notas UML



La Figura 4.4 ilustra algunos de estos conceptos. En la Figura 4.4a, desde una *ASPX Server Page a* es posible navegar a una de las páginas retornadas por la *Action r* definida en el *Controller s*, por medio de una solicitud GET desde la página fuente a la acción. La solicitud se representa con la asociación estereotipada con *Action Link* que define el valor etiquetado *Parameters* para representar el objeto de transferencia de datos que tranfiere los datos desde la página hasta la acción. *Action r* ejecuta lógica de aplicación, posiblemente involucrando la capa de negocio, y retorna una de las paginas *ASPX Server Page b* o *ASPX Server Page c* de acuerdo a la condición booleana. Este comportamiento se representa por la asociación estereotipada con *Return* que define el valor etiquetado *Condition*. La asociación *Return* también define el valor etiquetado *View Model* para

representar los datos construidos por la lógica de aplicación y que pueden formar parte de la página destino.

En la Figura 4.4b, desde una *ASPX Server Page a*, que contiene un *ASPX Form f*, es posible navegar, por medio de una solicitud POST, a una de las páginas retornadas por la *Action r* definida en el *Controller s*. La solicitud se representa con la asociación estereotipada con *Action Submit* que define el valor etiquetado *Parameters* para representar el objeto de transferencia de datos que transfiere los datos del formulario desde la página hasta la acción. *Action r* ejecuta lógica de aplicación, posiblemente involucrando la capa de negocio, y retorna una de las paginas *ASPX Server Page b* o *ASPX Server Page c* de acuerdo a la condición booleana. Este comportamiento se representa con la asociación estereotipada con *Return* que define el valor etiquetado *Condition*. La asociación *Return* también define el valor etiquetado *View Model* para representar los datos construidos por la lógica de aplicación y que pueden formar parte de la página destino.

4.4.3 Transformación WAE4.NET a código

El framework ASP.NET MVC proporciona componentes de vinculación automática que, usando un conjunto de reglas integradas, tienen la capacidad de traducir parámetros de tipo clave-valor en objetos, conocidos como *Input Model*, que son usados como parámetros en los métodos de los controladores. De igual forma el framework permite referenciar objetos, conocidos como *View Models*, dentro de la definición XML de las páginas (conocidas como *View Templates*) para presentar el contenido dinámico. En ASP.NET MVC, la sintaxis de las *View Templates* y dónde deben estar localizadas depende del componente responsable de construir el código HTML listo para ser enviado a los clientes Web (conocidos como *View Engines*). El framework incorpora dos tipos de *View Engine* nativos: ASPX (Reilly, 2006) y Razor (Chadwick, 2011). El presente trabajo ha seleccionado generar el código de acuerdo al *View Engine* ASPX debido a que es el *View Engine* ofrecido por defecto por el framework y está disponible en todas sus versiones.

Como en el caso del perfil WAE4JSF, el presente trabajo ha desarrollado un conjunto de reglas de transformación modelo a texto (M2T) que permiten transformar los modelos WAE4.NET en código ASP.NET MVC. La Tabla 4.51 especifica estas reglas de transformación explícitas que posibilitan la transformación manual y que pueden ser implementadas en cualquier motor de transformación. Para demostrar esta característica,

al igual que con WAE4JSF, el presente trabajo ha implementado estas reglas conforme al motor de transformaciones M2T implementado en Borland Together. El Anexo 8.2 muestra la implementación de algunas de estas reglas, las cuales, son específicas de Borland Together y se definen en términos de transformaciones XSLT que usan expresiones OCL para seleccionar elementos de los modelos y generar ficheros de texto a partir de ellos que constituyen el código de las aplicaciones ASP.NET MVC.

Por medio de la ejecución de las reglas de transformación, los modelos WAE4.NET se transforman en: (i) componentes ASP.NET MVC relacionados con la capa de presentación Web, incluyendo páginas ASPX, clases controlador con sus respectivas acciones, objetos de transferencia de datos conocidos como *Input Models* y *View Models*; (ii) el control de acceso basado en roles (RBAC) a nivel de acciones implementado como atributos de las acciones; (iii) clases .NET que se corresponden con las clases UML estándar incluidas en los modelos; y (iv) la estructura de directorios que el *View Engine* ASPX impone para encontrar cada fichero que define las páginas, y el cual debe ser replicado en los entornos de producción. Por supuesto, las páginas ASPX deben complementarse con el contenido que define la estructura de presentación y las clases pertenecientes a las capas de negocio e integración deben implementarse con el código de la aplicación, pero el esquema de las páginas ASPX, incluyendo formularios, botones de comandos, enlaces (*links*) y controladores con sus respectivas acciones se generan por la aplicación de estas reglas.

Tabla 4.51: Transformación WAE4.NET a componentes ASP.NET MVC

Estereotipo WAE4.NET	**Componente ASP.NET MVC**	**Descripción**
ASPX Server Page	Página ASPX	Página ASPX que se corresponde con el componente *View Template*. Éste describe una página Web usando la sintaxis del *View Engine* ASPX y se localiza de acuerdo a la estructura de directorios que impone este *View Engine*
ASPX Form	Componente ASP.NET MVC Form	El componente Form representa formularios definidos en una página ASPX usando la declaración Html.BeginForm(). Este componente es capaz de enviar sus datos a una acción especificada

Estereotipo WAE4.NET	**Componente ASP.NET MVC**	**Descripción**
Controller	Clase controlador	La clase controlador generada hereda de la clase base *Controller* definida en el framework y se localiza de acuerdo a la estructura de directorios impuesta por el *View Engine* ASPX
Action	Método público definido en una clase controlador, decorada con el atributo *ActionName*	El método público generado define los parámetros de entrada para recibir los datos de la petición, ejecuta lógica de aplicación involucrando la capa de negocio para generar los datos de la página siguiente y retorna un objeto *ActionResult*. El *View Engine* utiliza este objeto para localizar la próxima vista y combinarla con los datos generados, construyendo la próxima página en el proceso navegacional. El atributo *ActionName* permite desacoplar el nombre del método del nombre de la acción
Asociación dirigida *Action Link* desde un *ASPX Server Page* a un *Action*	Componente *Link* dentro de una página ASPX capaz de ejecutar una petición tipo GET. Parámetro de entrada en el método público del controlador	El componente Link se define dentro de la página ASPX usando la declaración Html.ActionLink(), el cual define los parámetros action y controller. El *Action* destino de la asociación se usa para construir el contenido del primer parámetro, y el *Controller* contenedor del *Action* se usa para construir el contenido del segundo parámetro. El valor etiquetado *Parameters* definido en el estereotipo *Action Link* se usa para construir el parámetro de entrada del método generado

Estereotipo WAE4.NET	**Componente ASP.NET MVC**	**Descripción**
Asociación dirigida *Action Submit* desde un *ASPX Form* a un *Action*	Componente Form capaz de ejecutar una petición tipo POST Parámetro de entrada en el método público del controlador	La declaración Html.BeginForm() define los parámetros action y controller. El *Action* destino de la asociación se usa para construir el contenido del primer parámetro, y el *Controller* contenedor del *Action* se usa para construir el contenido del segundo parámetro. El valor etiquetado *Parameters* definido en el estereotipo *Action Submit* se usa para construir el parámetro de entrada del método generado
Asociación dirigida *Return* desde un *Action* a un *ASPX Server Page*	Descripción del cuerpo del método público construido a partir del elemento *Action*	Esta asociación significa que el método público retorna la página ASPX como siguiente página en el proceso navegacional. Este comportamiento se codifica dentro del método público usando un objeto View Model cuyo valor se construye a partir del valor etiquetado *View Model* definido en el estereotipo Return y el *ASPX Server Page* destino de la asociación
Composición entre *Master Page* y *TCLPlace Holder*/*TCFPlace Holder*	Página *Master Page* capaz de especificar regiones con contenido por defecto a través del elemento asp:PlaceHolder. Cada *TCLiteral/TCFragment* representa una región con contenido por defecto	El elemento asp:PlaceHolder define las regiones en la *Master Page*. El contenido de este elemento se crea a partir del valor etiquetado *Text* o *URL* de los componentes *TCLiteral/TCFragment* respectivamente.
Asociación dirigida *Template Instance* entre un *Template Instantation* y un *Master Page*	Página ASPX configurada con el atributo MasterPageFile igual al nombre de la *Master*	El atributo MasterPageFile permite cargar el contenido de una página Master Page.

Estereotipo WAE4.NET	**Componente ASP.NET MVC**	**Descripción**
	Page destino de la asociación	
Dependencia entre un *TCILPlace Holder/TCIPlace Holder* a un *TCLPlace Holder/TCFPlace Holder*	Elemento asp:Content en la página ASPX que contiene el *TCILPlace Holder/TCIPlace Holder*	El elemento asp:Content permite sobrescribir una región definida en una *Master Page* para agregar contenido propio. Este elemento define el atributo ContentPlaceHolderId que especifica el nombre de la región a sobrescribir y cuyo contenido se crea a partir del *TCLPlace Holder/TCFPlace Holder* destino de la dependencia. El contenido del elemento asp:Content se crea a partir del *TCILPlace Holder/TCIPlace Holder* fuente de la dependencia
Asociación dirigida *Has Permission* entre un *Role* y un *Action*	Atributo Authorize en el método público Clase controlador llamado Authorization con los métodos LogOn y LogOff Sección Authentication en el fichero web.config	El método público que se construye a partir del *Action* destino de la asociación se decora con el atributo Authorize. Este atributo define el parámetro Role, el cual se construye a partir del *Role* fuente de la asociación, denotando que sólo los usuarios con el rol especificado pueden ejecutar el método decorado. El controlador Authorization se genera con dos métodos. El método LogOn procesa las credenciales recibidas y decide retornar la página de Login o la página solicitada, mientras que el método LogOff desconecta al usuario actual y redirige a la página Login. La sección Authentication en el fichero web.config especifica el mecanismo de autenticación basado en formularios. La aplicación redirecciona a los usuarios a una URL especificada cada vez que un usuario sin autenticar intenta ejecutar una acción decorada con el atributo Authorize

4.5 Comparación de WAE4JSF y WAE4.NET

Respecto a los frameworks considerados, aunque JSF y ASP.NET MVC integran el patrón de diseño MVC en su arquitectura, desde el punto de vista navegacional existe una diferencia fundamental: JSF es un framework basado en componentes, mientras que ASP.NET MVC es un framework basado en acciones. En JSF cada petición Web se dirige a una página o a un managed bean, y en ASP.NET MVC cada petición Web resulta en la ejecución de una acción. Los meta-modelos desarrollados en el presente trabajo, tienen en cuenta esta diferencia. De acuerdo al meta-modelo WAE4JSF, la navegación puede especificarse directamente entre páginas (lo que se ha llamado navegación estática) o por medio de un managed bean (lo que se ha llamado navegación dinámica), mientras de acuerdo al meta-modelo WAE4.NET, la navegación debe especificarse siempre por medio de acciones, las cuales son métodos públicos definidos en una clase controlador. Por tanto, sólo se soporta la navegación dinámica.

Por tanto, WAE4JSF ofrece más flexibilidad para especificar la navegación en una aplicación Web. Sin embargo, los componentes WAE4JSF deben traducirse a diferentes componentes de código dependiendo de si están modelando navegación estática o dinámica. Por el contrario, los componentes WAE4.NET son siempre traducidos unívocamente a los componentes de código facilitando la trazabilidad de la transformación de modelos a código. Este método es menos flexible pero más simple.

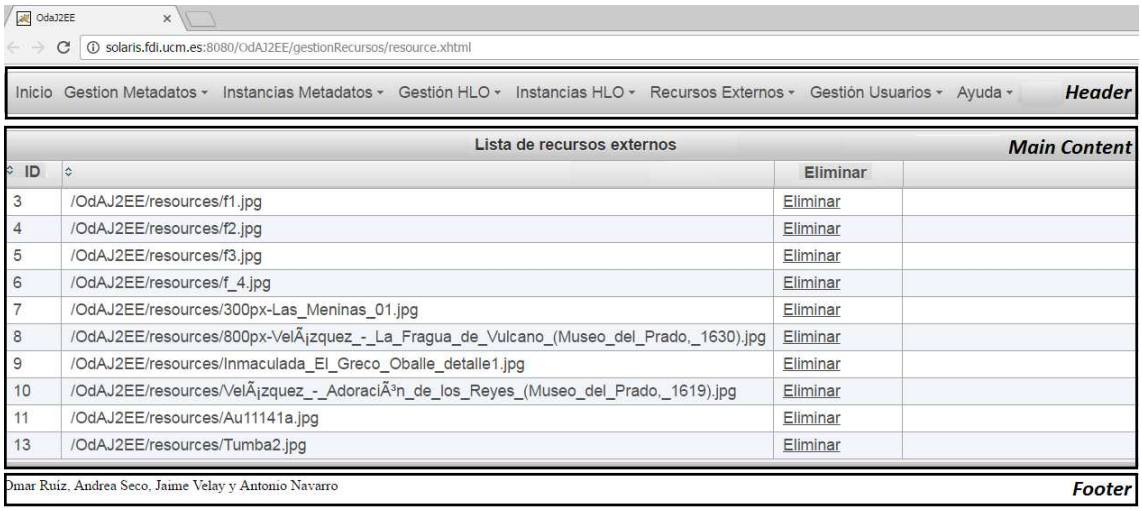
Respecto al modelado del control de acceso basado en roles (RBAC), los modelos WAE4JSF y WAE4.NET son equivalentes. En WAE4JSF las características RBAC se aplican directamente a las páginas Web, generando código declarativo basado en el framework JAAS en un solo fichero de configuración. Por tanto, este mecanismo sigue apropiadamente los principios de encapsulación y cohesión. En WAE4.NET, las características RBAC se aplican declarativamente a las acciones generando un atributo en la definición de cada acción, y por tanto, generando atributos de código a lo largo de toda la aplicación donde las acciones sean implementadas. Finalmente, la definición de la estructura de la interfaz de usuario a través de plantillas es equivalente en JSF y en WAE4.NET MVC, tanto a nivel de modelado como de código.

4.6 Ejemplo de aplicación de WAE4x: OdAJ2EE

Para facilitar la ilustración del proceso de desarrollo de la aplicación, en esta sección se modela con los perfiles WAE4x el mismo caso de uso utilizado con NMMp en el capítulo anterior. La Figura 4.5 muestra una captura de pantalla de la aplicación que implementa

el caso de uso que permite la eliminación de entidades *Recurso* gestionadas por la aplicación, el cual sólo pueden ejecutar los administradores de la aplicación. Como se puede ver, la estructura de la aplicación está basada en una plantilla con tres regiones llamadas: *Header*, *Main Content* y *Footer*. La página mostrada en la figura muestra una tabla de recursos con un enlace en cada uno de ellos que proporciona los medios para eliminarlo. Si un administrador selecciona uno de ellos, la aplicación navega a una página de confirmación que muestra los detalles del recurso seleccionado y pide confirmación para eliminarlo. Si el administrador confirma la eliminación del recurso, la aplicación intenta eliminarlo, mostrando una página informativa en caso de éxito o una página de error en caso contrario. Las próximas secciones muestran como se modela este caso de uso con los perfiles WAE4x, así como una evaluación de la experiencia de desarrollo.

Figura 4.5: Captura de pantalla OdAJ2EE para eliminación de recursos.



4.6.1 OdAJ2EE con WAE4JSF

La Figura 4.6 caracteriza el caso de uso que define la eliminación de recursos en términos de un modelo UML de clases extendido con el perfil WAE4JSF. El diagrama define cuatro páginas *JSF: ResourcesPage, DeletingResourcePage, DeletedResourcePage y ErrorPage*. La página *ResourcesPage* carga y muestra los recursos actuales organizados de forma tabular y permitiendo eliminar cada uno de ellos. Si se intenta eliminar un recurso, el usuario es redirigido a la página *DeletingResourcePage*, la cual muestra los detalles del recurso seleccionado y ofrece la opción de eliminarlo definitivamente. Si se decide eliminar el recurso y el proceso de eliminación se ejecuta correctamente, el usuario es redirigido a la página *DeletedResourcePage*, en caso de error, el usuario es redirigido a la página *ErrorPage*.

Los elementos *Template Instantiation*, los cuales son un tipo especial de *JSF Server Page*, tienen la capacidad de instanciar una plantilla para incluir el contenido definido en sus regiones y además la capacidad de sobrescribir algunas regiones para añadir su propio contenido. Como muestra la Figura 4.6, todas las páginas representadas por los elementos *Template Instantiation*, instancian la plantilla llamada *OdATemplate*, reutilizando el contenido definido en las regiones *Header* y *Footer*. Además todos los elementos *Template Instantiation* definen una región que sobrescribe la región *MainContent*, y por tanto, incluyen su propio contenido ahí. Nótese que la región *Header* contiene un *Static Link* que da acceso a la página *ResourcesPage* a través del outcome *ToResources*. Dado que *Static Link* representa navegación estática que no requiere de la ejecución de lógica de aplicación para navegar a la página destino, esto significa que los usuarios pueden navegar siempre a la página *ResourcesPage* desde cualquier página que instancie la plantilla llamada *OdATemplate*.

La página *ResourcesPage*, define la región *Resources* que sobrescribe la región *MainContent* de la plantilla. Esta región carga y presenta los recursos actuales. Para cargar los recursos usa el Managed Bean *ResourcesBean*, el cual depende de la interfaz *ResourceService* para obtener los recursos actuales. Esta interfaz puede implementarse con un servicio de aplicación local (*Application Service*), un delegado de negocio (*Business Delegate*) que accede a un servicio remoto o cualquier otro patrón multicapa (Alur et al., 2003) que implemente la lógica de negocio relacionada con la gestión de recursos. Para presentar los recursos en la interfaz de usuario, la región *Resources* define el elemento *ResourcesTable*, el cual tiene la capacidad de presentar los datos en forma tabular. Nótese que este componente contiene la columna *DeleteColumn* que permite navegar a la página *DeletingResourcePage* a través del outcome *ToDeletingResource* enviándole el identificador del recurso seleccionado.

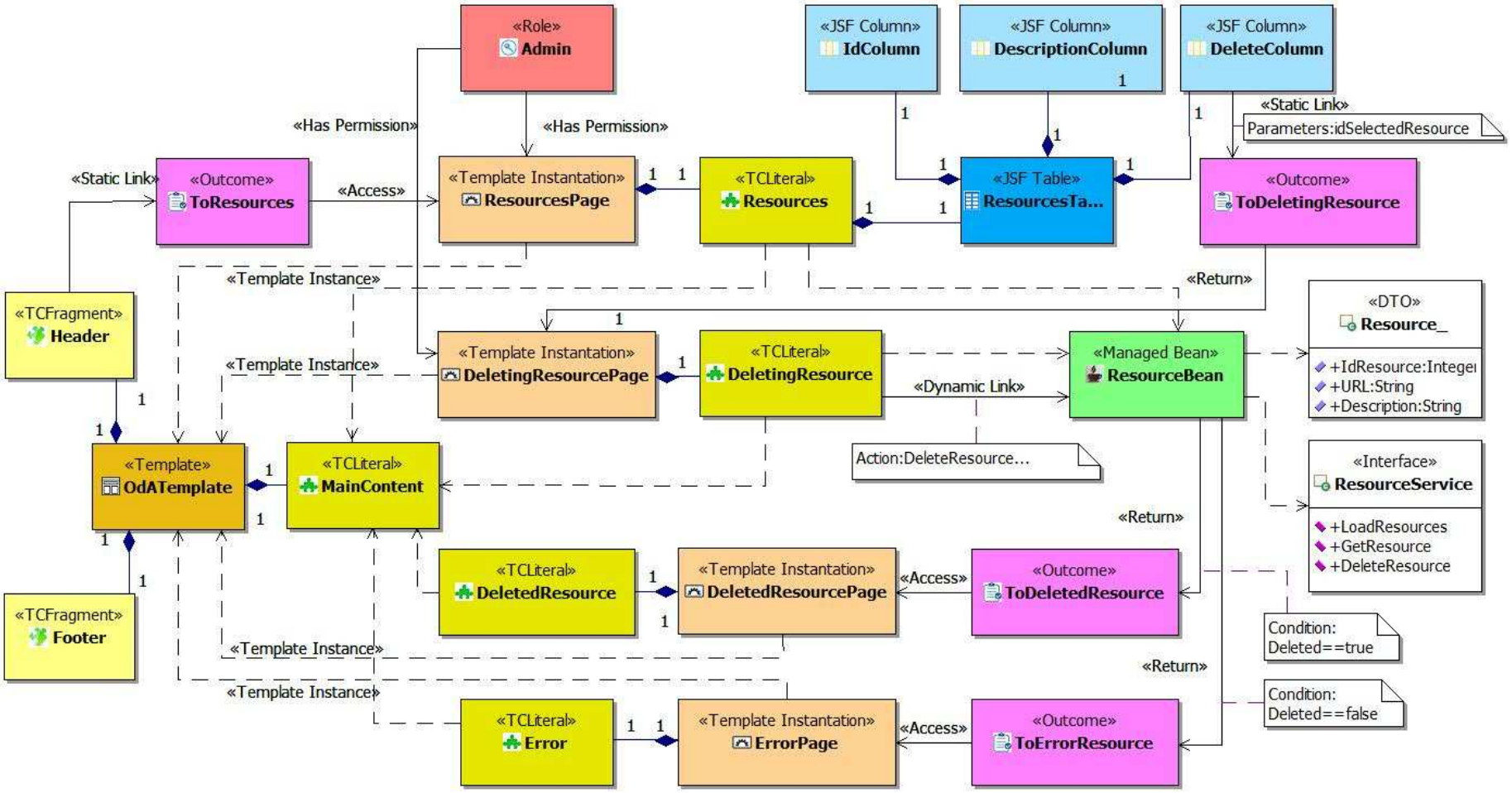
La página *DeletingResourcePage* actúa como página de confirmación de borrado. La región que define utiliza el elemento managed bean para cargar y presentar los detalles del recurso seleccionado. Además, esta región permite navegar dinámicamente a una de las páginas, *DeletedResourcePage* o *ErrorPage* dependiendo del outcome retornado por la ejecución del método *DeleteResource* definido en el managed bean, el cual delega otra vez en la interfaz *ResourceService* la ejecución de las reglas de negocio para eliminar los recursos. Este paso navegacional se modela usando la asociación estereotipada con *Dynamic Link* entre la región *DeletingResource* y el managed bean *ResourceBean*. La asociación *Dynamic Link* define el valor etiquetado *Action* que contiene el nombre del

método a invocar en el managed bean y el valor etiquetado *Parameters* que contiene el identificador del recurso a eliminar.

Por otro lado, como muestra el diagrama, sólo los usuarios incluidos en un rol llamado *Admin* pueden acceder a las páginas protegidas. Cuando los usuarios solicitan una de estas páginas, se carga un formulario pidiendo sus credenciales. Este mecanismo de control de acceso basado en roles (RBAC) se implementa usando las capacidades ofrecidas por el framework JAAS, el cual sólo requiere una sección en el fichero de configuración declarando los roles y sus relaciones con las páginas que forman parte de la aplicación.

Nótese que este modelo navegacional puede incluir otras clases de presentación JSF, e incluir clases UML pertenecientes a otras capas de la aplicación. Estas clases, y sus correspondientes diagramas, no se incluyen en el ejemplo con el fin de mantener la sencillez del modelo de presentación, pero la interfaz *ResourceService* representa el punto de entrada a ellos.

Figura 4.6: Eliminación de recursos en OdAJ2EE con WAE4JSF



4.6.2 OdAJ2EE con WAE4.NET

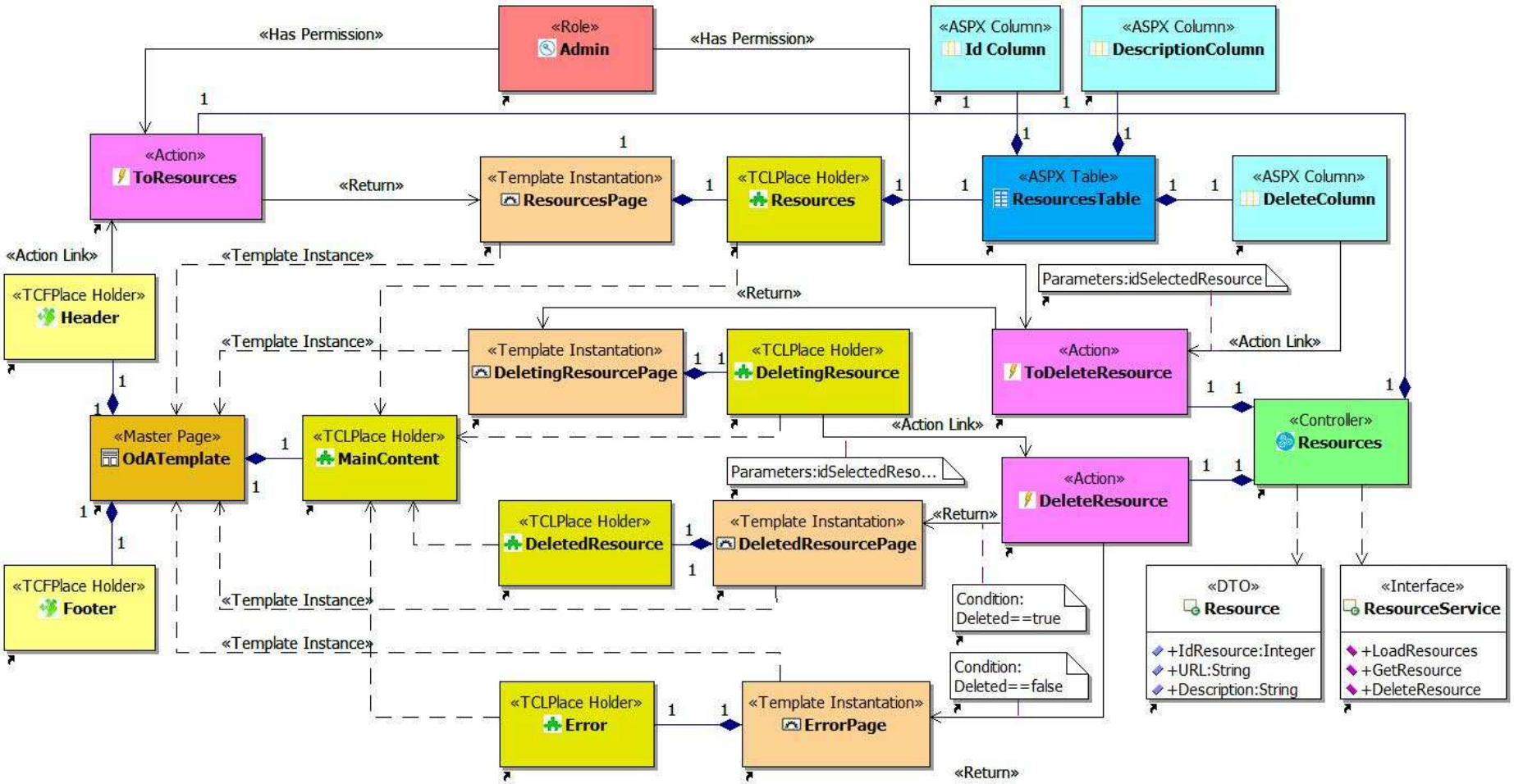
La Figura 4.7 caracteriza el caso de uso que define la eliminación de recursos en términos de un modelo UML de clases extendido con el perfil WAE4.NET. Todos los elementos *Template Instantiation*, que son un tipo especial de *ASPX Server Page* con la capacidad de instanciar plantillas y sus regiones, instancian la misma plantilla, que en el contexto de ASP.NET MVC se conoce como *Master Page*. En el ejemplo, La *Master Page* define tres regiones, que en el contexto de ASP.NET MVC se conocen como *Place Holder*: *Header*, *Footer* y *MainContent*. Dado que WAE4JSF y WAE4.NET aprovechan las características similares que los frameworks JSF y ASP.NET MVC ofrecen para implementar las plantillas y sus regiones, la descripción de estos componentes hecha en la sección anterior puede aplicarse, traduciendo los nombres al modelo WAE4.NET.

El *Place Holder* llamado *Header* define una asociación estereotipada con *Action Link*, la cual lanza una solicitud a la acción *ToResources* definida en el controlador *Resources*. Esta acción ejecuta lógica de aplicación, involucrando la capa de negocio a través de la interfaz *ResourceService*, para construir un objeto de transferencia de datos (conocido como *View Model*) con la información de los recursos actuales, y retorna la página *ResourcesPage*, la cual presenta los recursos en forma tabular y permite la eliminación de cada uno. La columna *DeleteColumn* definida en la tabla de recursos permite lanzar una solicitud a la acción *ToDeleteResource* definida en el controlador *Resources* enviándole el identificador del recurso seleccionado. Esta acción obtiene y compone un *View Model* con la información detallada del recurso seleccionado involucrando la capa de negocio y retorna la página *DeletingResourcePage*, la cual presenta esta información y pide confirmación para eliminar el recurso.

Para eliminar definitivamente el recurso, la página *DeletingResourcePage* permite lanzar una petición a la acción *DeleteResource* definida en el controlador *Resources* enviándole el identificador del recurso a eliminar. Esta acción intenta eliminar el recurso involucrando la capa de negocio a través de la interfaz *ResourceService*, y retorna la página *DeletedResourcePage* o *ErrorPage* de acuerdo al resultado de la operación.

Por otro lado, como el diagrama muestra, sólo los usuarios incluidos en un rol llamado *Admin* pueden ejecutar las acciones protegidas y, por tanto, acceder a las páginas retornadas por ellas. El framework ASP.NET MVC redirige automáticamente a los usuarios a una acción llamada *LogOn* definida en un controlador llamado *Authorization* cada vez que intentan ejecutar una acción protegida.

Figura 4.7: Eliminación de recursos en OdAJ2EE con WAE4.NET



4.6.3 Discusión del modelado de OdAJ2EE

Para probar la aplicabilidad de los perfiles WAE4x, se decidió utilizarlos en el desarrollo de la aplicación OdAJ2EE[9]. OdAJ2EE es una aplicación Java compleja, con alrededor de 100 páginas Web dinámicamente construidas y un esquema de persistencia basado en JPA. Los perfiles WAE4x, debido a que están basados en UML, pueden desplegarse y utilizarse en herramientas UML CASE genéricas. El presente trabajo los ha implementado en la herramienta CASE Borland Together 2008, la cual, junto con Eclipse, se usó a lo largo de todo el proceso de desarrollo de OdAJ2EE. Debido a que OdAJ2EE está basada en la arquitectura multicapa, cada capa pudo ser desarrollada independientemente y en cualquier orden. Este trabajo decidió diseñar cada capa como un paquete UML diferente, comenzando con la capa de negocio, la capa de integración y finalmente la capa de presentación.

En las capas de negocio e integración, se utilizó UML estándar para especificar su estructura y comportamiento. Se utilizaron diagramas de clases UML para definir los objetos de negocio y los servicios de aplicación y diagramas de secuencia UML para especificar su comportamiento. De esta forma, los patrones arquitectónicos y de diseño pudieron agregarse explícitamente en los modelos UML. Respecto a la capa de presentación, dado que OdAJ2EE se implementó en J2EE, cada caso de uso se especificó a través de un flujo navegacional modelado con un diagrama WAE4JSF, resultando un total de 48 diagramas de este tipo. Aunque no se desarrollaron los diagramas equivalentes con WAE4.NET, no hay diferencias significativas entre los dos perfiles, más allá, por supuesto, de las impuestas por los respectivos frameworks.

De esta forma, los componentes de los modelos WAE4JSF pudieron ser fácilmente integrados con los componentes de la capa de negocio y de la capa de integración siguiendo los patrones multicapa. Por tanto el perfil WAE4JSF nos permitió modelar la capa de presentación utilizando componentes implementados en frameworks específicos, que de otra forma no podrían haberse hecho explícitos, y aprovechar la expresividad de UML para modelar la integración de la capa de presentación con el resto de capas siguiendo patrones multicapa.

[9] http://solaris.fdi.ucm.es:8080/OdAJ2EE/

La expresividad de los perfiles WAE4x quedó demostrada porque sólo con los elementos definidos en cada perfil se pudo de desarrollar mapas navegacionales de cualquier complejidad. Los elementos definidos en cada perfil fueron intuitivos de usar, o al menos muy fácil de aprender para cualquier desarrollador con conocimientos de los frameworks JSF o ASP.NET MVC. Los modelos WAE4x fueron concebidos como una colección de páginas Web conectadas en un mapa a través de componentes definidos en frameworks específicos que indicaban el camino navegacional de una página a la siguiente. Los componentes definidos en los frameworks permitieron modelar el flujo navegacional sin ambigüedad a través de la integración con los componentes de la capa de negocio y la especificación de condiciones booleanas alcanzadas por estos. Las páginas Web fueron concebidas como contenedores que definen los componentes capaces de lanzar procesos navegacionales y en los cuales se debe insertar los controles de interfaz de usuario adicionales y definir la estructura de presentación más adelante en el proceso de desarrollo.

El proceso de desarrollo de los mapas navegacionales con los perfiles WAE4x permitió abstraerse de las complejidades de los frameworks y código, centrándose en elementos gráficos y sus relaciones. En este sentido, los perfiles se consideraron como una nueva capa de abstracción para especificar flujos navegacionales de forma rápida y flexible. El control de acceso basado en roles (RBAC) y la definición de la estructura de la interfaz de usuario a través de plantillas se incluyeron en el modelo de forma sencilla, ya que fue suficiente con definir las relaciones entre roles, plantillas y páginas.

También se comprobó en OdAJ2EE la escalabilidad de los perfiles WAE4x, ya que la aplicación define cerca de 100 páginas dinámicas. Como en el caso de UML, los modelos WAE4x pudieron incluir páginas y componentes agrupados funcionalmente en diferentes paquetes y por tanto definir enlaces navegacionales entre paquetes. De esta forma, los modelos navegacionales se mantuvieron en un tamaño razonable, abriendo la posibilidad de crear a partir de ellos la estructura de directorios con el código de la aplicación.

5 ENTERPRISE WAE

5.1 Introduccíon

Como se comentó en el Capítulo 4, los perfiles WAE4x se han desarrollado con el objetivo de soportar el modelado de la capa de presentación Web de aplicaciones empresariales modernas. Estos perfiles extienden la definición de UML-WAE para caracterizar los componentes de la capa de presentación implementados en los frameworks JSF y ASP.NET MVC, dos de los más ampliamente utilizados en la industria. De esta forma, los perfiles WAE4x permiten modelar la capa de presentación Web utilizando componentes específicos cuya semántica se define a través de reglas de transformación a los componentes implementados en los frameworks. Esta semántica facilita leer los modelos pensando en términos de los correspondientes componentes y además evita la ambigüedad en el modelado. Por tanto, las reglas de transformación definen una especificación ejecutable de los modelos, clarificando su transformación a código, y, en última instancia, cerrando la brecha entre modelos y código.

Los modelos generados con WAE4x son modelos específicos de la plataforma (PSMs), que al igual que los modelos UML, modelan código directamente en cualquier herramienta UML CASE genérica. Por tanto, la generación y mantenimiento del código no depende de herramientas específicas. Así modelos y código pueden evolucionar independientemente, sin necesidad de herramientas propietarias que los relacione. Sin embargo, esta característica hace que los modelos WAE4x tengan un nivel de abstracción bajo y por tanto, pueden convertirse rápidamente en modelos complejos con un nivel de detalle demasiado elevado.

El fin último del presente trabajo es facilitar el desarrollo de software a través de modelos (en línea con el desarrollo dirigido por modelos) que permitan generar código. Pero, para facilitar el desarrollo se requiere la definición de modelos simples con alto nivel de abstracción. Además, para generar código mantenible, se requiere que la semántica de cada concepto en los modelos este bien definida en términos de reglas que permitan su transformación a modelos más detallados o a código de manera independiente de herramientas específicas (*black-box wizards*). Por tanto, en el contexto de este trabajo, se requiere elevar el nivel de abstracción de los modelos WAE4x, preservando una semántica consistente para facilitar su usabilidad práctica.

La experiencia con NMMp demostró que el uso de modelos independientes de plataformas (PIMs) es muy valioso en el proceso de desarrollo porque permiten expresar los mapas navegacionales con un alto nivel de abstracción haciendo que los modelos sean sencillos y muestren una visión general de la aplicación. Por tanto, NMMp fue el primer candidato a evaluar como notación fuente a partir de la cual generar, por medio de reglas de transformación explícitas, modelos WAE4x. Sin embargo, dado que NMMp define la estructura navegacional en términos de transiciones de un página a otra por medio de anclas de recuperación (*retrieval anchors*) y anclas computacionales (*computing anchors*), que abstraen los componentes del lado del servidor, obliga a que estos sean generados en el proceso de trasformación dificultando su trazabilidad.

Para salvar este obstáculo, el presente trabajo define *Enterprise WAE* (E-WAE), una extensión UML que, preservando todos los beneficios de NMMp, añade un nivel de abstracción sobre los perfiles WAE4x. E-WAE permite caracterizar mapas navegacionales utilizando conceptos propios de aplicaciones empresariales modernas, como ejecución de procesos computacionales, objetos de transferencia de datos, componentes que representan plantillas y sus regiones y elementos de control de acceso basado en roles (RBAC), entre otros, manteniendo un nivel alto de abstracción. De esta forma los modelos E-WAE pueden ser vistos como diseños de alto nivel de una aplicación, mientras los modelos WAE4x como diseños detallados de la misma aplicación. Además, el hecho de que E-WAE y WAE4x definan una semántica consistente y que todos los perfiles UML comparten un núcleo semántico común (OMG, 2013), facilita la definición de reglas de transformación bien definidas entre estos dos tipos de modelos.

Por tanto, desde el punto de vista de la arquitectura dirigida por modelos, E-WAE permite generar modelos independientes de la plataforma para representar la capa de presentación

Web obviando detalles arquitectónicos y conceptos de programación, mostrando así una visión general de la aplicación que facilita la comunicación entre usuarios finales y desarrolladores. Además, el presente trabajo define un conjunto de reglas explícitas que permiten la transformación de los modelos E-WAE en modelos WAE4x, los cuales son modelos específicos de la plataforma que proporcionan una semántica orientada a código de los modelos E-WAE. Como se describió en el Capítulo 4, el presente trabajo también define un conjunto de reglas explícitas que permiten transformar los modelos WAE4x en código de frameworks específicos.

Para demostrar la aplicabilidad de E-WAE, al igual que con NMMp, se decidió desplegar los perfiles E-WAE y WAE4x e implementar las reglas de transformación explícitas en la herramienta CASE Borland Together 2008 (ahora parte de MicroFocus), por incluir no solo un entorno para la definición de perfiles UML sino todo un conjunto de capacidades MDA basadas en los estándares de modelado OCL, QVT y MOF. Con el despliegue de los perfiles y la implementación de las reglas de transformación, el presente trabajo permite la transformación automática de los modelos E-WAE en modelos WAE4x, y estos en código listo para ser ejecutado en sus respectivos IDE: Eclipse para las aplicaciones JSF y Microsoft Visual Studio para las aplicaciones ASP.NET MVC. Este código de la capa de presentación listo para ser ejecutado puede ser usado como mockups interactivos que permite la validación anticipada de la capa de presentación Web. Con esta característica, E-WAE permite facilitar la etapa de obtención y formulación de requisitos, los cuales constituyen la base para la planificación de cualquier proyecto de software, su diseño y su posterior desarrollo (Rivero et al., 2013).

En las metodologías de desarrollo tradicional como cascada o incremental, los requisitos se obtienen y formulan siguiendo metodologías y documentación bien definidas. Sin embargo, en el contexto de las aplicaciones empresariales, esta etapa es con frecuencia de difícil ejecución. Entre las causas que la dificultan están la volatilidad de los requisitos causada principalmente por la naturaleza dinámica de las demandas actuales, la falta de entendimiento de sus propias demandas por parte de los interesados y la falta de habilidades y/o vocabulario técnico para comunicarlas (Thakurta, R., & Ahlemann, F., 2011).

Por otro lado, las metodologías de desarrollo ágil, como Scrum o eXtreme Programming son adecuadas en el desarrollo de aplicaciones en las cuáles los requisitos no están bien definidos, son difíciles de especificar o pueden cambiar rápidamente, incluso durante el desarrollo (Stober, T., & Hansmann, U., 2010). Sin embargo, actualmente, estas

metodologías resultan de difícil aplicación en el desarrollo de aplicaciones empresariales, ya que este tipo de aplicaciones necesitan un diseño arquitectónico y documentación bien definidos que aseguren su efectividad y mantenibilidad. Por otro lado, si existe un diseño arquitectónico predefinido, la integración continua sin la documentación adecuada puede fácilmente degradarlo (Sommerville, 2010).

En cualquier caso, tanto en las metodologías de desarrollo tradicional, como en las metodologías de desarrollo ágil, el uso de mockups de presentación ha demostrado ser una herramienta muy efectiva en la etapa de obtención y formulación de requisitos, debido a que son técnicamente valiosos para los desarrolladores y al mismo tiempo, completamente entendibles para los usuarios finales, permitiendo una comunicación sin ambigüedad entre ellos (Ricca, F. et al., 2010).

Los mockups de presentación se han convertido en un excelente punto de partida para el desarrollo de software, ya que por medio de la estructura de la interfaz de usuario describen todas las funcionalidades soportadas por la aplicación. Si los mockups son interactivos, es decir, soportan la interacción de los usuarios finales, permiten además, validar la capa de presentación en el campo, observar a los usuarios finales en acción, analizar su comportamiento en el tiempo y escuchar sus opiniones (Esposito, 2016). Los beneficios de esta clase de mockups han sido reconocidos tanto en la academia como en la industria (Ricca, F. et al., 2010; Esposito, 2016).

Sin embargo, generalmente, los mockups interactivos se utilizan sólo para refinar los requisitos por medio de la demostración de características específicas a los usuarios. Estos mockups son rápidamente desarrollados sin tener en cuenta los requisitos no funcionales ni el diseño arquitectónico, y por tanto no con la idea de ser reutilizados en el proceso de desarrollo de las aplicaciones finales. Generalmente, los desarrolladores desechan los mockups y empiezan el proceso de desarrollo implementado los requisitos identificados (Rivero et al., 2010).

Por medio de los mockups interactivos generados por el presente trabajo, que han sido llamados *Mockups Navegacionales*, E-WAE incorpora algunas ventajas de las metodologías ágiles en el proceso de desarrollo, sin abandonar las características de las metodologías tradicionales. Como las metodologías ágiles, E-WAE promueve el desarrollo a través de ciclos iterativos cortos centrados en el desarrollo de la capa de presentación, involucrando a los usuarios finales en cada iteración. Como en los procesos tradicionales, E-WAE usa la notación UML para definir mapas navegacionales, los cuales

actúan como modelos fuente a partir de los cuales se generan los mockups navegacionales siguiendo la filosofía MDD. Por tanto, los mockups están soportados por modelos bien definidos que aseguran la calidad y legibilidad suficiente para ser completamente reutilizados en la aplicación final.

De hecho, no hay mucha investigación y literatura acerca de la aplicación de los procesos ágiles en el desarrollo de aplicaciones empresariales (Breivold et al., 2010; Buschmann, F., & Henney, K., 2013). El diseño de la arquitectura implica invertir esfuerzo y tiempo suficiente para definir qué componentes forman el sistema, cómo se relacionan entre ellos y cómo mediante su interacción cumplen con los requisitos funcionales especificados, cumpliendo además con los requisitos no funcionales como rendimiento, seguridad, robustez, disponibilidad o usabilidad. Sin embargo, este diseño de arquitectura, y su posterior mantenimiento, causan un conflicto con los procesos ágiles que promueven las iteraciones de desarrollo rápidas para mantener la alta interacción con los usuarios finales.

Por tanto, el enfoque presentado puede ser visto como un intento de reconciliar el diseño de la arquitectura y los procesos ágiles. Desarrolladores y usuarios finales pueden trabajar juntos para descubrir requisitos funcionales a través de los mapas navegacionales y los mockups interactivos generados a partir de ellos. Además, debido a que los mapas navegacionales son modelos UML, pueden ser utilizados como documentación bien definida y a la vez, pueden ser fácilmente extendidos con modelos que incluyan patrones arquitectónicos y de diseño generando aplicaciones que implementan una arquitectura definida por los desarrolladores.

Dado que los perfiles E-WAE y WAE4x pueden ser desplegados en cualquier herramienta CASE UML y las transformaciones PIM a PSM y PSM a código están claramente definidas, el mantenimiento de código no depende de herramientas específicas. Además, dado que los frameworks JSF y ASP.NET MVC has sido diseñados para desarrollar la capa de presentación Web de aplicaciones empresariales, si se utiliza los modelos E-WAE y WAE4x para modelar la capa de presentación y UML plano para modelar el resto de capas, es posible modelar una aplicación empresarial completa por medio de modelos legibles a los programadores, delegando en ellos el desarrollo del software y permitiendo la libre inclusión de patrones arquitectónicos y de diseño suficientemente probados en la industria (Alur et al., 2003; Fowler, 2002).

Respecto al desarrollo de la interfaz de usuario (por ejemplo labels, textbox, combobox, etc.), E-WAE puede ser usado desde etapas tempranas de desarrollo, por ejemplo, como

herramienta de soporte en el proceso de obtención y formulación de requisitos. Por tanto, se centra en capturar el comportamiento funcional de la capa de presentación, la información relacionada al control de acceso basado en roles (RBAC), los datos demandados en la capa de presentación y la estructura navegacional de las aplicaciones. De esta forma, E-WAE promueve la construcción rápida de la interfaz de usuario evitando los detalles de *look & feel* y otros detalles no relevantes en las etapas tempranas de desarrollo.

5.2 Desarrollo del Perfil E-WAE

Como se ha comentado, los perfiles WAE4x son una buena solución para modelar la capa de presentación Web de aplicaciones empresariales modernas. Los perfiles WAE4x proporcionan una extensión de UML que caracteriza los componentes de capa de presentación implementados en frameworks empresariales, y permiten que estos se integren con los modelos UML estándar que representan al resto de capas de la aplicación. Por tanto, los perfiles permiten: (i) caracterizar la capa de presentación Web de aplicaciones empresariales desarrolladas con JSF o ASP.NET MVC usando perfiles UML a partir de los requisitos de las aplicaciones; (ii) mejorar el desarrollo de código y la mantenibilidad a través de la inclusión explicita de patrones arquitectónicos y de diseño en los modelos UML; (iii) incluir el control de acceso basado en roles (RBAC) en los modelos; y (iv) soportar la implementación de reglas de transformación automáticas que permitan la generación de código desde los modelos dando total control sobre su generación.

Sin embargo, dado que los modelos WAE4x tienen nivel de abstracción bajo y, por tanto, un nivel de detalle adecuado solo para desarrolladores, E-WAE añade un nivel de abstracción sobre ellos por medio de la definición de estereotipos que: (i) caracterizan la capa de presentación Web utilizando conceptos propios de aplicaciones empresariales modernas (como la ejecución de procesos computacionales, objetos de transferencia de datos, definición de plantillas, etc.), pero independientemente de las plataformas de implementación subyacentes; (ii) permiten perfilar los datos demandados en la capa de presentación; (iii) incluyen en los modelos información relacionada al control de acceso basado en roles (RBAC) de manera independiente de la tecnología; y (iv) pueden ser transformados manual o automáticamente en modelos WAE4x, los cuales pueden ser transformados automáticamente a código que puede ser utilizado como mockups interactivos para ser validados por los usuarios finales. Por otro lado, dado que los

modelos WAE4x son modelos PSM basados en UML, estos son legibles a los desarrolladores, pudiendo ser enriquecidos e integrados por ellos a través de la implementación explicita de patrones arquitectónicos y de diseño. Por tanto, los mockups navegacionales automáticamente generados desde los modelos WAE4x pueden ser completamente reutilizados en el desarrollo de aplicaciones Web con arquitecturas concretas

Aunque las reglas de transformación explícitas desarrolladas en el presente trabajo permiten que los modelos E-WAE puedan ser transformados manualmente en modelos WAE4x, evitando la dependencia en herramientas específicas (*black-box wizards*), estas reglas también asisten y promueven su implementación en motores de transformación automáticos que habiliten el desarrollo a través de ciclos iterativos cortos, como las metodologías ágiles proponen. De hecho, el presente trabajo ha implementado las reglas bien definidas utilizando el lenguaje estándar QVT *Operational Mappings* (OMG,2008) implementado por la herramienta CASE Borland Together para transformar automáticamente los modelos E-WAE en modelos WAE4x de acuerdo al estándar OMG.

5.2.1 Meta-modelo MOF para E-WAE

Como en el caso de NMMp y WAE4x, el presente trabajo define un meta-modelo MOF para proporcionar especificaciones precisas de los elementos definidos en el perfil E-WAE. La Figura 5.1 muestra este meta-modelo. El elemento *Page* caracteriza las páginas procesadas por el servidor antes de ser enviadas al cliente. E-WAE no considera páginas directamente enviadas sin procesamiento, debido a que estas no son promovidas por los frameworks empresariales modernos (Cortés & Navarro, 2016). La estructura navegacional se modela usando los elementos *Link* y *Submit*. El elemento *Link* representa una asociación dirigida desde un elemento *PageFragment* o *Column* hasta un elemento *Page* o *Function*. Cuando el destino de la asociación es un elemento *Page*, significa que el proceso navegacional no requiere la ejecución de lógica de aplicación, por tanto, modela la navegación estática. Cuando el destino de la asociación es un elemento *Function*, significa que el proceso navegacional requiere ejecución de lógica de aplicación para encontrar la siguiente página de acuerdo a condiciones booleanas representadas por elementos *ReturnCondition*, modelando por tanto la navegación dinámica. En ambos casos, la navegación se ejecuta por medio de una petición de tipo GET que puede transportar parámetros representado por los elementos *GETParam*.

De igual forma, los elementos *Submit* representan la navegación dinámica desde un elemento *Form* contenido en un elemento *PageFragment* a un elemento *Function*. En este caso la navegación se ejecuta por medio de una solicitud de tipo POST que transporta los datos del formulario en elementos *DataCarrier*. De hecho, los elementos *PageFragment* y *Form* pueden usar elementos *DataCarrier* para representar los datos que se usan y transportan entre la interfaz de usuario y los componentes tras de la capa de presentación. Estos pueden representar estructuras de datos de complejidad arbitraria incluyendo tipos primitivos, tipos anidados y colecciones, permitiendo perfilar los objetos de transferencia de datos que pueden usarse para mover información entre las capas de una arquitectura multicapa (Alur et al., 2003; Fowler, 2002).

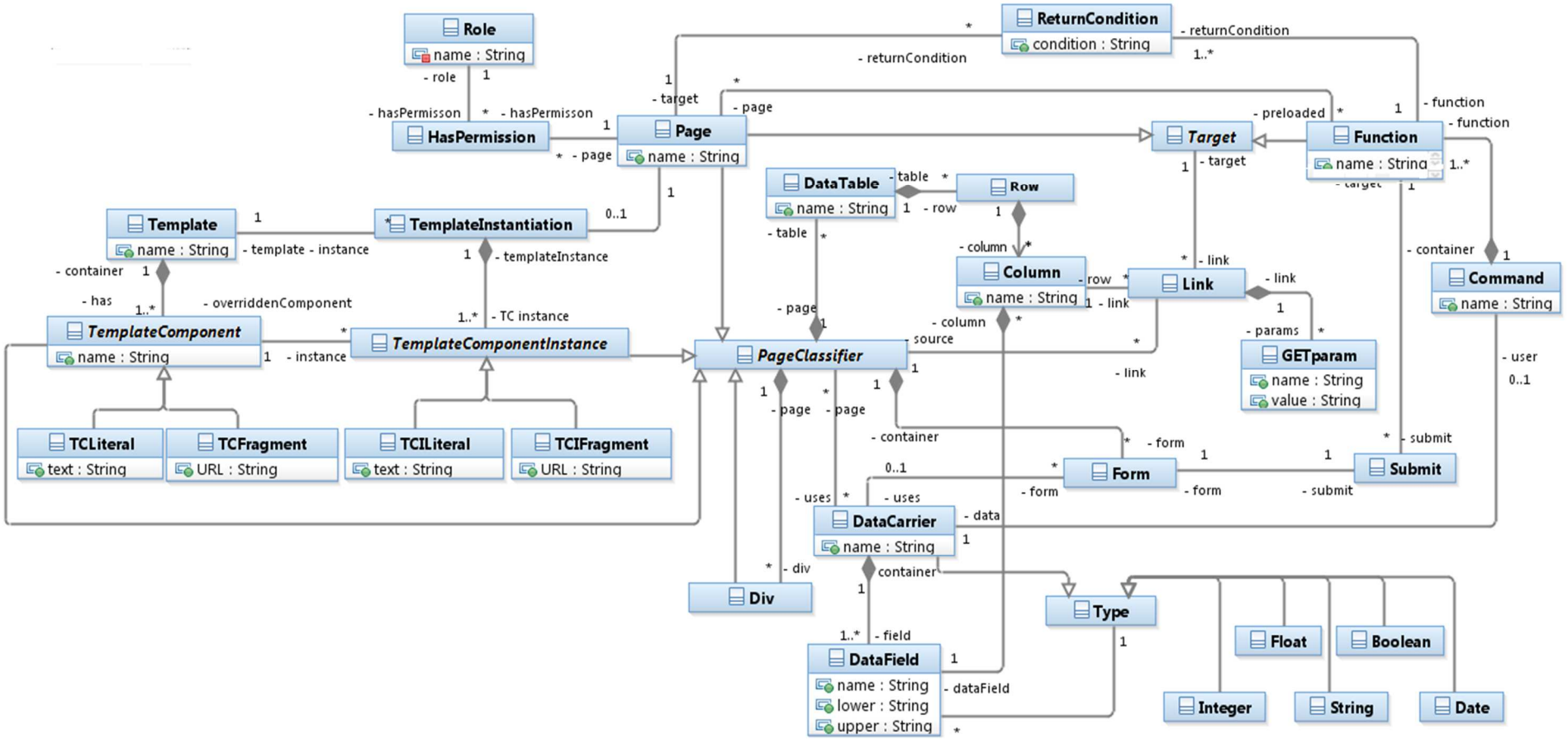
Como la Figura muestra, los elementos *PageFragment* pueden contener elementos *DataTable* y *Div*. Los elementos *DataTable* representan los componentes ampliamente usados en las capas de presentación Web para administrar colecciones de datos. Estos elementos están compuestos de elementos *Row* que representan cada ítem de la colección y de elementos *Column*, los cuales conteniendo un elemento *DataField* representan las propiedades de los ítems. Además los elementos *Column* pueden relacionarse con elementos *Link* indicando que ellos también pueden lanzar procesos navegacionales. Los elementos *Div* representan divisiones o secciones definidas dentro de elementos *PageFragment* que a su vez comparten su misma estructura y comportamiento.

La definición del diseño y comportamiento común a través de plantillas se modela usando los elementos *Template* y *TemplateComponent*. Estos elementos caracterizan plantillas y sus regiones respectivamente, definiendo el contenido común que puede ser literal (representado por el elemento *TCLiteral*) o cargado usando una URL (representado por el elemento *TCFragment*). Los elementos *Page* pueden instanciar una plantilla por medio del elemento *TemplateInstantiation* reutilizando el contenido común y con la capacidad de sobrescribir las regiones a través de elementos *TemplateComponentInstance*, los cuales permiten agregar su propio contenido que también puede ser literal (representado por el elemento *TCILiteral*) o cargado usando una URL (representado por el elemento *TCIFragment*).

Los elementos *Function* se definen dentro de elementos *Command*, de forma que pueden ser vistos como métodos definidos en una clase. La información relacionada con el control de acceso basado en roles (RBAC) se representa con el elemento *Has Permission* que relaciona a los elementos *Role* y *Page* especificando qué roles en el dominio de la aplicación tienen permisos sobre qué páginas.

Por otro lado, como su nombre sugiere, E-WAE está influenciado por UML-WAE, sin embargo en la práctica son notaciones muy diferentes debido a que: (i) E-WAE no considera el concepto de *Client Page* definido en UML-WAE; (ii) los enlaces navegacionales en E-WAE se definen en términos de componentes implementados en frameworks empresariales, mientras que en UML-WAE se definen en términos de anclas HTML; (iii) el diseño y comportamiento común en E-WAE se define en términos de plantillas y regiones implementadas en frameworks empresariales, mientras que en UML-WAE se definen en términos de marcos (frames) HTML; y (iv) E-WAE soporta el modelado del control de acceso basado en roles (RBAC), los cuales no son considerados en UML-WAE. Estas diferencias hacen posible la transformación de los modelos PIM E-WAE en modelos PSM WAE4x a través de reglas de transformación bien definidas que pueden ser ejecutadas manual o automáticamente.

Figura 5.1: Meta-modelo MOF para E-WAE



5.2.2 Perfil UML para E-WAE

Aunque el meta-modelo MOF para E-WAE es muy descriptivo, el presente trabajo establece la semántica de los conceptos especificados en él en términos de un perfil UML con el objetivo de hacerlo usable en herramientas UML CASE genéricas. Las Tablas 5.1 a 5.28 definen las reglas de traducción entre los conceptos del meta-modelo y los estereotipos y valores etiquetados que los representan en el perfil UML.

Tabla 5.1: Concepto E-WAE Page Classifier

Concepto E-WAE	**Page Classifier**
Estereotipo	-
Elemento UML	-
Descripción	Clasificador abstracto que define la estructura y comportamiento común de los elementos *Page*, *Div*, *TemplateComponent* y *TemplateComponentInstance.*
Valores etiquetados	-

Tabla 5.2: Estereotipo E-WAE Page

Concepto E-WAE	**Page**
Estereotipo	Page
Elemento UML	Clase
Icono	
Descripción	Páginas procesadas por el servidor antes de ser enviadas al cliente.
Valores etiquetados	-

Tabla 5.3: Estereotipo E-WAE Form

Concepto E-WAE	**Form**
Estereotipo	Form
Elemento UML	Clase
Icono	

Descripción	Formularios definidos dentro de elementos *Page Classifier.*
Valores etiquetados	-

Tabla 5.4: Estereotipo E-WAE DataCarrier

Concepto E-WAE	**DataCarrier**
Estereotipo	DataCarrier
Elemento UML	Clase
Icono	
Descripción	Estructuras de datos de cualquier complejidad definidas en elementos *Page Classifier* o *Form* que pueden ser accesibles desde los elementos *Command.*
Valores etiquetados	-

Tabla 5.5: Estereotipo E-WAE DataField

Concepto E-WAE	**DataField**
Estereotipo	-
Elemento UML	-
Descripción	Descripción de las propiedades definidas en los elementos *DataCarrier* incluyendo sus nombres y tipos.
Valores etiquetados	-

Tabla 5.6: Concepto E-WAE Type

Concepto E-WAE	**Type**
Estereotipo	-
Elemento UML	-
Descripción	Tipos UML definidos para los elementos *DataField.*
Valores etiquetados	-

Tabla 5.7: Concepto E-WAE Target

Concepto E-WAE	**Target**
Estereotipo	-
Elemento UML	-
Descripción	Clasificador abstracto para elementos *Page* y *Function* que actúan como destinos de enlaces navegacionales.
Valores etiquetados	-

Tabla 5.8: Estereotipo E-WAE Function

Concepto E-WAE	**Function**
Estereotipo	Function
Elemento UML	Clase
Icono	
Descripción	Ejecución de procesos computacionales que ejecutan lógica de la aplicación para encontrar la próxima página en un proceso computacional.
Valores etiquetados	-

Tabla 5.9: Estereotipo E-WAE Command

Concepto E-WAE	**Command**
Estereotipo	Command
Elemento UML	Clase
Icono	
Descripción	Contenedor de elementos *Function* y *DataCarrier*.
Valores etiquetados	-

Tabla 5.10: Estereotipo E-WAE ReturnCondition

Concepto E-WAE	**ReturnCondition**
Estereotipo	Return

Elemento UML	Asociación dirigida
Icono	-
Descripción	Navegación a una página de acuerdo a condiciones booleanas alcanzadas por la ejecución de lógica de la aplicación representada por elementos *Function.*
Valores etiquetados	*Condition*: condición booleana alcanzada por la ejecución de lógica de aplicación.

Tabla 5.11: Estereotipo E-WAE Preload

Concepto E-WAE	**Asociación entre *Page* y *Function***
Estereotipo	Preload
Elemento UML	Dependencia
Descripción	Invocación desde una página a un proceso computacional antes de que se construya la parte visual de la página.
Valores etiquetados	-

Tabla 5.12: Estereotipo E-WAE Link

Concepto E-WAE	**Link**
Estereotipo	Link
Elemento UML	Asociación dirigida
Descripción	Relación navegacional por medio de una solicitud de tipo GET desde un elemento *Page Classifier* o *Column* a una página (navegación estática) o a un proceso computacional (navegación dinámica).
Valores etiquetados	*Parameters*. Usado para mover datos entre las paginas y procesos computacionales.

Tabla 5.13: Estereotipo E-WAE Submit

Concepto E-WAE	**Submit**
Estereotipo	Submit
Elemento UML	Asociación dirigida
Icono	-

Descripción	Un formulario que envía sus datos a un proceso computacional modelando la navegación dinámica. La navegación se ejecuta a través del método POST.
Valores etiquetados	*Parameters*: parámetros transportados que se corresponden con los datos del formulario

Tabla 5.14: Concepto E-WAE GETParam

Concepto E-WAE	**GETParam**
Estereotipo	-
Elemento UML	-
Descripción	Parámetros enviados en una solicitud que se traducen en el valor etiquetado *Parameters* definido en los elementos que representan relaciones navegacionales estáticas o dinámicas.
Valores etiquetados	-

Tabla 5.15: Estereotipo E-WAE Template

Concepto E-WAE	**Template**
Estereotipo	Template
Elemento UML	Clase
Icono	
Descripción	Página base o plantilla que define un diseño y un comportamiento común reutilizable por otras páginas en una aplicación.
Valores etiquetados	-

Tabla 5.16: Estereotipo E-WAE TCLiteral

Concepto E-WAE	**TCLiteral**
Estereotipo	TCLiteral
Elemento UML	Clase
Icono	

Descripción	Región definida dentro de una plantilla. Su contenido es completamente definido dentro del elemento.
Valores etiquetados	*Text*. Usado para describir su contenido.

Tabla 5.17: Estereotipo E-WAE TCFragment

Concepto E-WAE	**TCFragment**
Estereotipo	TCFragment
Elemento UML	Clase
Icono	
Descripción	Región definida dentro de una plantilla. Su contenido se carga desde elementos externos.
Valores etiquetados	*URL*. Usado para describir el recurso desde donde se carga su contenido.

Tabla 5.18: Estereotipo E-WAE TemplateInstantiation

Concepto E-WAE	**TemplateInstantiation**
Estereotipo	TemplateInstantiation
Elemento UML	Clase
Icono	
Descripción	Componente capaz de instanciar una plantilla permitiendo reutilizar sus regiones y sobrescribir algunas de ellas.
Valores etiquetados	-

Tabla 5.19: Estereotipo E-WAE TCILiteral

Concepto E-WAE	**TCILiteral**
Estereotipo	TCILiteral
Elemento UML	Clase
Icono	

Descripción	Región que, definida en una página que instancia una plantilla, sobrescribe a una región definida en la plantilla. El contenido que define su estructura y comportamiento se define completamente dentro del elemento.
Valores etiquetados	*Text*. Usado para describir su contenido.

Tabla 5.20: Estereotipo E-WAE TCIFragment

Concepto E-WAE	**TCIFragment**
Estereotipo	TCIFragment
Elemento UML	Clase
Icono	
Descripción	Región que, definida en una página que instancia una plantilla, sobrescribe a una región definida en la plantilla. Su contenido se carga desde elementos externos.
Valores etiquetados	*URL*. Usado para describir el recurso desde donde se carga su contenido.

Tabla 5.21: Estereotipo E-WAE TemplateInstance

Concepto E-WAE	**Asociación dirigida entre *TemplateInstantiation* y *Template***
Estereotipo	TemplateInstance
Elemento UML	Dependencia
Descripción	Componente *TemplateInstantiation* que instancia una plantilla para reutilizar el contenido y comportamiento definido en sus regiones.
Valores etiquetados	-

Tabla 5.22: Estereotipo E-WAE DataTable

Concepto E-WAE	**DataTable**
Estereotipo	DataTable
Elemento UML	Clase

Icono	
Descripción	Componente extensivamente usado en interfaces Web para administrar datos tabulares.
Valores etiquetados	-

Tabla 5.23: Estereotipo E-WAE Row

Concepto E-WAE	**Row**
Estereotipo	Row
Elemento UML	Clase
Descripción	Componente que representa un ítem en los datos tabulares.
Valores etiquetados	-

Tabla 5.24: Estereotipo E-WAE Column

Concepto E-WAE	**Column**
Estereotipo	Column
Elemento UML	Clase
Icono	
Descripción	Componente que representa un campo de datos de un ítem en los datos tabulares. Además este elemento permite lanzar un proceso navegacional.
Valores etiquetados	-

Tabla 5.25: Estereotipo E-WAE Div

Concepto E-WAE	**Div**
Estereotipo	Div
Elemento UML	Clase
Icono	

Descripción	Sección o división definida dentro un *Page Classifier* que además comparte su estructura y comportamiento.
Valores etiquetados	-

Tabla 5.26: Estereotipo E-WAE Role

Concepto E-WAE	**Role**
Estereotipo	Role
Elemento UML	Clase
Icono	
Descripción	Conjunto de roles definidos en el dominio de la aplicación.
Valores etiquetados	-

Tabla 5.27: Estereotipo E-WAE Has Permission

Concepto E-WAE	**Has Permission**
Estereotipo	Has Permission
Elemento UML	Asociación dirigida
Descripción	Relación dirigida entre un rol y una página indicando que el rol tiene permisos sobre la página.
Valores etiquetados	-

5.2.3 Transformación E-WAE a WAE4x

Como se ha comentado, los modelos E-WAE pueden ser considerados como modelos de alto nivel de los modelos WAE4x. Por tanto los modelos E-WAE se pueden considerar como diseños abstractos de una aplicación Web (PIMs) mientras los modelos WAE4x como diseños concretos de la misma aplicación (PSMs). Además el hecho de que todos los perfiles UML compartan un núcleo semántico común facilita la definición de reglas de transformación entre modelos extendidos con perfiles. Con este concepto en mente, el presente trabajo ha definido un conjunto de reglas para transformar modelos E-WAE en modelos WAE4x. Las reglas de transformación se definen en la Tabla 5.28 y son la base para el desarrollo de las transformaciones QVT *Operational Mappings* que se han

implementado para permitir la transformación automática siguiendo la filosofía MDD y de acuerdo con los estándares OMG. Los Anexos 8.3 y 8.4 muestran la implementación de algunas de estas reglas. Las reglas de transformación ilustradas en la Tabla 5.28, básicamente transforman:

- Componentes E-WAE de alto nivel tales como Page, Form, Template y Template Component en componentes específicos de las plataformas tales como: (i) JSF Server Page, JSF Form, Template y TemplateComponent; y (ii) ASPX Server Page, ASPX Form, Master Page y Place Holder.
- Componentes E-WAE que representan ejecución de lógica de aplicación ocultando conceptos de programación tales como *Function* y *Command* en componentes de plataforma especifica relacionados con la ejecución de código, tales como: (i) *JSF Managed Bean* y el valor etiquetado *Action* definido en las relaciones navegacionales que modelas la navegación dinámica; y (ii) componentes *ASPX Controller* y *Action* que representan una clase controlador con sus métodos respectivamente.
- Relaciones navegacionales E-WAE como *Link* y *Submit* se descomponen en componentes que modelan la navegación estática cuando el destino de la relación es una página, o en navegación dinámica cuando el destino de la relación es un proceso computacional. Estas relaciones se transforman en: (i) componentes *JSF Static Link, Dinamic Link/Dynamic Submit, Access, Return y Outcome*; y (ii) *ASPX componentes Action Link/Action Submit y Return*.
- Componentes E-WAE como *DataCarrier* que representan datos contenidos en páginas o formularios, se traducen en valores etiquetados definidos en los estereotipos que representan relaciones navegacionales. Estos elementos pueden ser vistos como objetos de transferencia de datos que transfieren información entre la interfaz de usuario y otros componentes de plataforma específica de la capa de presentación Web.
- Componentes E-WAE relacionados con el control de acceso basado en roles (RBAC) se traducen en componentes que modelan las características RBAC a nivel de página en WAE4JSF y a nivel de acciones en WAE4.NET.

Tabla 5.28: Transformación E-WAE a WAE4x

Elemento E-WAE	**Elemento WAE4JSF**	**Elemento WAE4.NET**
Page	*JSF Server Page*	*ASPX Server Page*
Form	*JSF Form*	*ASPX Form*
Command	*Managed Bean*	*Controller*
Function	Valor etiquetado *Action* definido en los estereotipos *Dynamic Link* y *Dynamic Submit*	*Action*
DataTable	*JSF Data Table*	*ASPX Data Table*
Row	*JSF Row*	*ASPX Row*
Column	*JSF Column*	*ASPX Column*
Link desde *Page a* o *Div d* o *TemplateComponent t* o *Column c* hasta *page b*	*Static Link* desde *JSF Server Page a* o *JSF Div d* o *TemplateComponent t* o *JSF Column c* hasta *Outcome aTob* + *Access* desde *Outcome aTob* hasta *JSF Server Page b*	*Action Link* desde *ASPX Server Page a* o *ASPX Div d* o *Place Holder t* o *ASPX Column c* hasta *Action aTob* + C*ontroller* que compone *Action aTob* + *Return* con valor etiquetado C*ondition* igual a *true* desde *Action aTob* hasta *ASPX Server Page b*
Si *Page a* o *Div d* o *TemplateComponent t* usan un *DataCarrier*	Descripción de la clase *DataCarrier* incluyendo sus propiedades y tipos en el valor etiquetado *Parameters* definido en el estereotipo *Static Link*	Descripción de la clase *DataCarrier* incluyendo sus propiedades y tipos en el valor etiquetado *Input Model* del estereotipo *Action Link* + clase DTO (Data Transfer Object) con nombre y propiedades igual al *DataCarrier* + dependencia désde la clase *Controller* a la clase DTO
Link desde *Page a* o *Div d* o *TemplateComponent t* o *Column col* hasta *Function f* compuesta por *Command c*	*Dynamic Link* con valor etiquetado *Action* igual a *f* desde *JSF Server Page a* o *JSF Div d* o *Template Component t* o *JSF Column col* hasta *Managed Bean c*	*Action Link* desde *ASPX Server Page a* o *ASPX Div d* o *Place Holder t* o *ASPX Column col* hasta *Action f* compuesta por *Controller c*

Si *Page a* o *Div d* o *TemplateComponent t* usa un *DataCarrier*	Descripción de la clase *DataCarrier* incluyendo sus propiedades y tipos definidos en el *Managed Bean* y el valor etiquetado *Parameters* del estereotipo *Dynamic Link* + clase DTO con nombre y propiedades igual al *DataCarrier* + dependencia del *Managed Bean* al DTO	Descripción de la clase *DataCarrier* incluyendo sus propiedades y tipos dentro del valor etiquetado *Input Model* del estereotipo *Action Link* + clase DTO con nombre y propiedades igual al *DataCarrier* + Dependencia desde el controlador a la clase DTO
Return con valor etiquetado *Condition* igual *true* desde *Function f* compuesta por *Command c* hasta *Page b*	Return con valor etiquetado Condition igual true desde Managed Bean c hasta Outcome fTob + Access desde Outcome fTob hasta JSF Server Page b	*Return* con valor etiquetado *Condition* igual *true* desde *Action f* compuesta por *Controller c* hasta *ASPX Server Page b*
Submit desde *Form* a hasta *Function f* compuesta por *Command c*	Dynamic Submit con valor etiquetado Action f desde JSF Form a hasta Managed Bean c	Action Submit desde ASPX Form a hasta Action f compuesta por Controller c
Si *Form a* usa un *DataCarrier*	Descripción de la clase *DataCarrier* incluyendo sus propiedades y tipos en el *Managed Bean* y el valor etiquetado *Parameters* del estereotipo *Dynamic Submit* + clase DTO con nombre y propiedades igual al *DataCarrier* + Dependencia desde *Managed Bean* hasta clase DTO	Descripción de la clase *DataCarrier* incluyendo sus propiedades y tipos en el valor etiquetado *Input Model* del estereotipo *Action Submit* + clase DTO con nombre y propiedades igual al *DataCarrier* + Dependencia desde la clase *Controller* hasta la clase DTO
Template	*Template*	*Master Page*

Elemento E-WAE	**Elemento WAE4JSF**	**Elemento WAE4.NET**
TCLiteral	*TCLiteral*	*Literal Place Holder*
TCFragment	*TCFragment*	*Fragment Place Holder*
TemplateInstanciation	*TemplateInstanciation*	*TemplateInstanciation*
TCILiteral	*TCILiteral*	*ILiteral Place Holder*
TCIFragment	*TCIFragment*	*IFragment Place Holder*
TemplateInstance	*Template Instance*	*Template Instance*
Div	*JSF Div*	*ASPX Div*
Role	*Role*	*Role*
Has Permission desde *Role r* hasta *Page a*	*Has Permission* desde *Role r* hasta *JSF Server Page a*	*Has Permission* desde *Role r* hasta *Action* que retorna *ASPX Server Page a*

5.3 Ejemplos de aplicación de E-WAE

Para ilustrar el proceso de desarrollo con E-WAE, esta sección describe tres ejemplos de su aplicabilidad. El primer ejemplo está basado en el caso de uso de eliminación de entidades *Recurso* en la aplicación OdAJ2EE, el cual se ha utilizado a lo largo de esta memoria. En esta sección se desarrolla el modelo E-WAE que lo representa y que actúa como modelo fuente a partir del cual se generan los modelos WAE4x. También se describe la estructura de los proyectos y el código automáticamente generados a partir de estos modelos y que constituye los mockups navegacionales listos para ser ejecutados y validados por los usuarios finales. El segundo ejemplo está basado en el proceso de compra *on-line* implementado por la aplicación Web de Amazon, este ejemplo ilustra el desarrollo de los modelos E-WAE, su transformación a modelos WAE4x y finalmente la generación de los mockups. El tercer ejemplo está basado en la función de búsqueda de la Librería del Congreso de USA, el objetivo de este ejemplo es ilustrar la factibilidad de usar E-WAE para modelar menús desde la perspectiva navegacional.

5.3.1 Eliminación de Recursos en OdAJ2EE

En esta sección se ilustra la factibilidad de usar el perfil E-WAE en el desarrollo de la aplicación OdAJ2EE[10] (Navarro et al., 2013). Como se comentó en el Capítulo 4, OdAJ2EE es una aplicación Web compleja con alrededor de 100 páginas JSF, por tanto,

[10] Accesible en http://solaris.fdi.ucm.es:8080/OdAJ2EE

se construyeron modelos WAE4x para soportar su desarrollo y asegurar un mantenimiento apropiado.

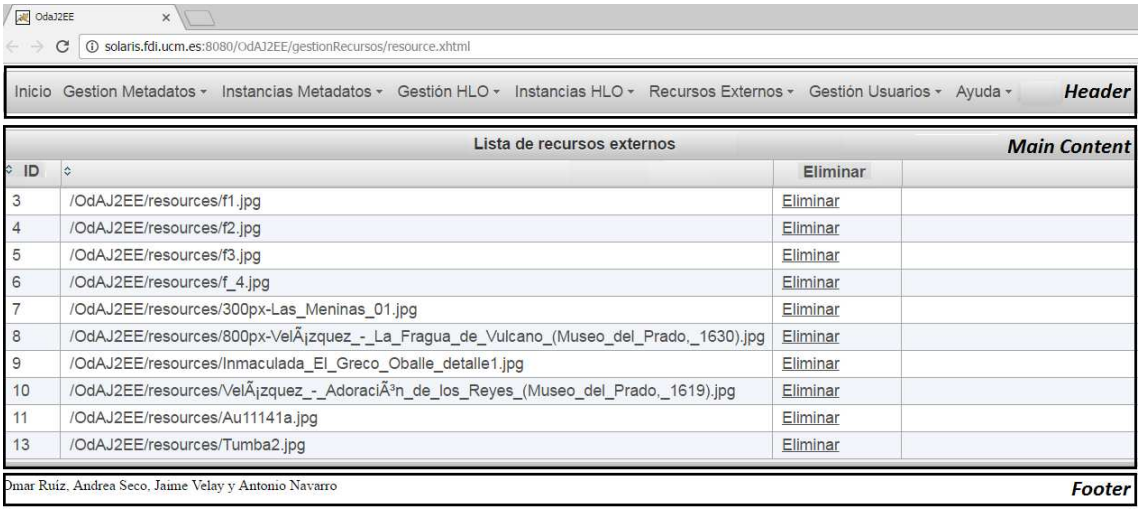
Para facilitar la ilustración del proceso E-WAE, en este capítulo se modela el mismo caso de uso modelado con NMMp y WAE4x en los Capítulos 3 y 4 respectivamente. Como se comentó, para añadir características de las metodologías ágiles al proceso de desarrollo, E-WAE requiere de la ejecución automática de las reglas de transformación de los modelos E-WAE a WAE4x y de estos a código. Por tanto, en esta fase del proyecto, el presente trabajo utiliza las reglas de transformación ilustradas en la Tabla 5.2 (E-WAE a WAE4x), en la Tabla 4.2 (WAE4JSF a código) y en la Tabla 4.4 (WAE4.NET a código) que se han implementado en la herramienta CASE Borland Together 2008. Las reglas de transformación de E-WAE a WAE4x se han implementado utilizando el lenguaje QVT-*Operational Mappings* (OMG, 2008), lo cual permite que puedan ejecutarse en cualquier herramienta que cumpla con este estándar OMG. Respecto a la generación de código, debido a la falta de soporte estándar para las transformaciones M2T en la mayoría de las herramientas UML CASE, se ha decido implementar las reglas de transformación de WAE4x a código en el motor de transformaciones M2T propio de Borland Together, por tanto, aunque solo pueden ser ejecutadas en esta herramienta, ilustran la factibilidad de su implementación. Desde el punto de vista de este trabajo, Borland Together proporciona un entorno de desarrollo integrado para todas sus necesidades, excepto por la pequeña aplicación que despliega el código generado en los respectivos IDEs. Sin embargo, excepto por las transformaciones automáticas M2T, el resto del enfoque puede ser soportado en cualquier herramienta UML CASE genérica.

El modelo E-WAE desarrollado se transforma automáticamente en los modelos WAE4x expuestos en las secciones 4.6.1 y 4.6.2 del capítulo anterior, los cuales a su vez se transforman automáticamente en código que implementa los mockup navegacionales listos para ser ejecutados y validados. De esta forma se ilustra el proceso E-WAE completo, desde el desarrollo de los modelos PIM, su transformación a modelos PSM y la generación de código que puede ser utilizado como mockups navegacionales listos para ser ejecutados y reutilizados en el proceso de desarrollo de la aplicación completa.

El caso de uso es la eliminación de entidades *Recurso* gestionadas por la aplicación, el cual puede ser ejecutado solo por los administradores. La Figura 5.2 muestra una captura de pantalla de la aplicación que implementa este caso de uso. Como se puede ver, la estructura de la aplicación está basada en una plantilla con tres regiones llamadas: *Header*, *Main Content* y *Footer*. La página mostrada en la figura contiene una tabla de

recursos con un enlace en cada uno de ellos que proporciona los medios para eliminarlo. Si un administrador selecciona en uno de ellos, la aplicación navega a una página de confirmación que muestra los detalles del recurso seleccionado y pide confirmación para eliminarlo. Si el administrador confirma la eliminación del recurso, la aplicación intenta eliminarlo mostrando una página informativa en caso de éxito o una página de error en caso contrario.

Figura 5.2: Captura de pantalla OdAJ2EE para eliminación de recursos.



5.3.1.1 Modelo E-WAE

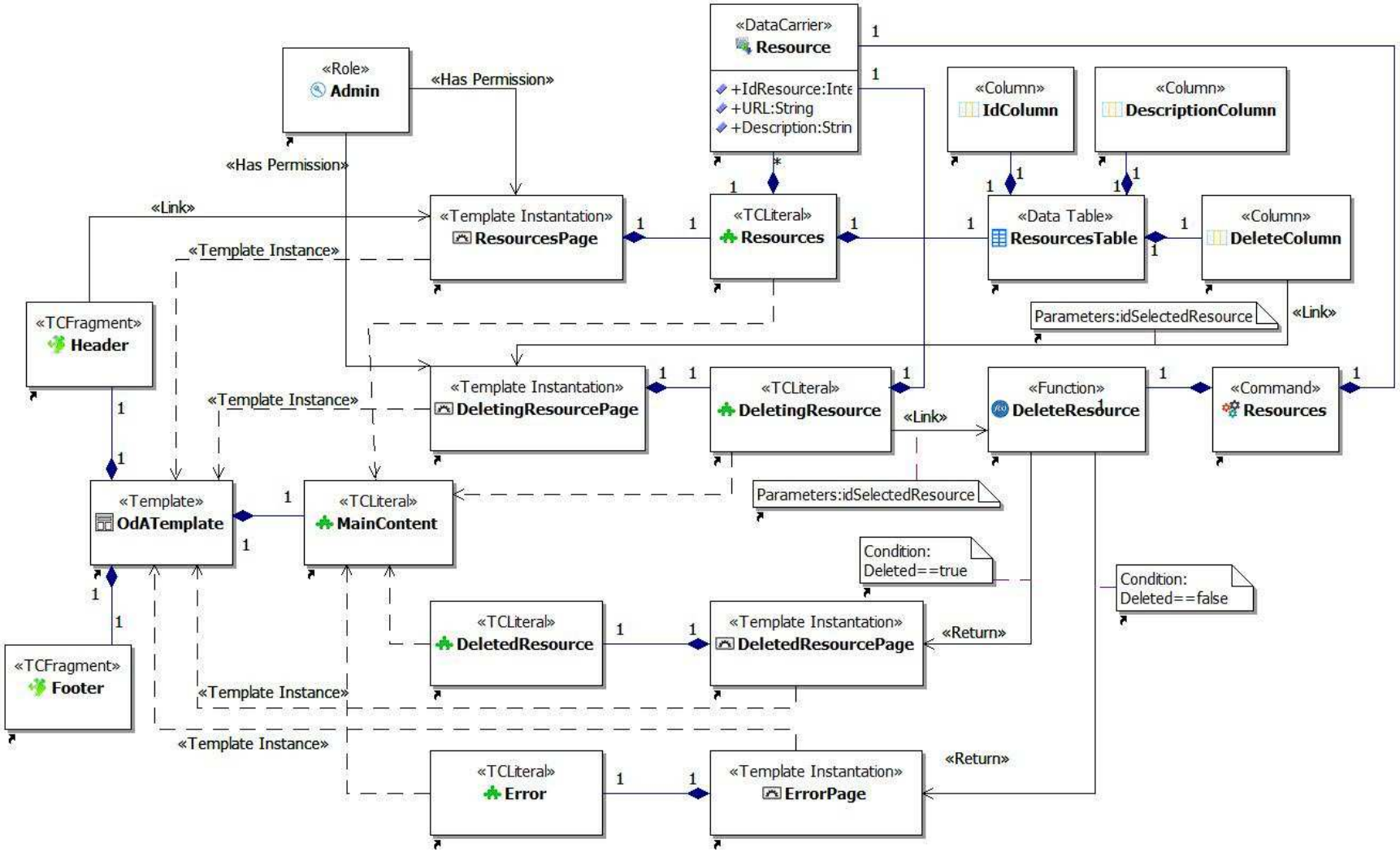
La Figura 5.3 caracteriza este caso de uso en términos de un diagrama de clases UML extendido con el perfil E-WAE. Como muestra la figura, la aplicación está basada en un elemento *Template*, el cual está compuesto de tres regiones llamadas *Header*, *Footer* y *MainContent*, que definen el diseño y comportamiento común. Los elementos *Template Instantiation*, los cuales son un tipo especial de elemento *Page*, tiene la capacidad de instanciar un elemento *Template* para incluir el contenido definido en sus regiones. Además, este elemento también puede sobrescribir las regiones definidas en el elemento *Template* para agregar su propio contenido.

Como se puede ver, todas las páginas, representadas por elementos *Template Instantiation*, instancian al elemento *Template* llamado *OdATemplate* reutilizando el contenido definido en las regiones *Header* y *Footer*, pero todos ellos definen su propia región que sobrescribe la región *MainContent*, y por tanto, incluyen ahí su propio contenido. Nótese que la región *Header* contiene un enlace navegacional que da acceso a la página *ResourcePage*. De esta forma, los usuarios siempre pueden navegar a la página

ResourcePage desde cualquier página que instancie la plantilla *OdATemplate* sin sobrescribir su región *Header*.

La región *Resources* definida en la página *ResourcesPage* está compuesta de una colección de entidades *Resource* (representadas por el elemento *DataCarrier*) y un elemento *DataTable* para mostrarlas en la interfaz de usuario. El elemento *DataTable* contiene la columna *DeleteColumn* que permite navegar a la página *DeletingResourcePage* enviándole el identificador del recurso seleccionado. La página *DeletingResourcePage*, que actúa como página de confirmación de borrado, muestra los detalles del recurso seleccionado en la región *DeletingResource* y permite lanzar un proceso navegacional enviando el identificador del recurso al elemento *Function DeleteResource*, el cual representa el proceso responsable de ejecutar la lógica de negocio necesaria para eliminar el recurso. La página destino depende de la condición booleana alcanzada por la ejecución de la lógica de negocio, la cual puede ser la página *DeletedResourcePage* si el proceso fue exitoso o la página *ErrorPage* en caso contrario. Por otro lado, como muestra la figura, el control de acceso basado en roles (RBAC) está representado por el elemento *Has Permission* que relaciona a los elementos *Role* y *Page*, especificando que sólo los usuarios incluidos en un rol llamado *Admin* pueden acceder a las páginas *ResourcesPage* y *DeletingResourcePage*

Figura 5.3: Eliminación de recursos en OdAJ2EE con E-WAE



5.3.1.2 Modelos WAE4x

Como se puede apreciar en la Figura 5.3, los diagramas E-WAE son modelos de diseño de alto nivel muy útiles para definir la estructura navegacional, la estructura de presentación a través de plantillas y el control de acceso basado en roles (RBAC) de las capas de presentación Web. Sin embargo, los modelos E-WAE están lejos del nivel de implementación donde puede incluirse los patrones arquitectónicos y de diseño. Por tanto, su transformación a modelos PSM basados en UML, tales como los modelos WAE4x es muy valiosa (Cortés & Navarro, 2016). Si se aplican las reglas de transformación de E-WAE a WAE4x definidas en la Tabla 5.4 al diagrama mostrado en la Figura 5.3, se obtienen los diagramas mostrados en las Figuras 4.6 y 4.7 del capítulo anterior. Cabe destacar que, aunque las reglas de transformación pueden aplicarse manualmente, estos diagramas se han generado automáticamente en la herramienta CASE Borland Together aplicando las transformaciones QVT *Operational Mappings* que implementan las reglas definidas en la Tabla 5.4.

Nótese como en las Figuras 4.6 y 4.7 han aparecido conceptos de implementación que no aparecían en la Figura 5.3. Por ejemplo, las asociaciones *Link* entre páginas de E-WAE se han transformado en asociaciones *Static Link* o *Action Link* que modelan la navegación estática. Las asociaciones *Submit* en E-WAE entre páginas y procesos computacionales se han transformado en *Dynamic Submit* o *Action Submit* que modelan la navegación dinámica. Los elementos *DataCarrier* de E-WAE definidos como parte de elementos de la interfaz de usuario se han transformado en objetos de transferencia de datos apropiadamente referenciados por componentes de la capa de presentación y serializados en los valores etiquetados *Parameters* o *Input Model* definidos en los enlaces navegacionales.

Además, han surgido componentes específicos de la plataforma como *Managed Beans, Outcomes, Controllers y Actions*. Por ejemplo, el elemento *Function* de E-WAE se ha transformado en el contenido del valor etiquetado *Action* definido en las asociaciones *Dynamic Link* o *Dynamic Submit* y usado para crear los elementos *Outcome* retornados por el *Managed Bean* y que dan acceso a la página destino. Por tanto, estos elementos *Function* de E-WAE ocultan los conceptos y componentes presentes en los modelos WAE4x (o cualquier otra notación PSM para la cual las transformaciones hayan sido definidas) que implementan la navegación.

Nótese que los modelos WAE4x obtenidos se pueden optimizar o enriquicer manualmente con otros componentes definidos en la plataforma específica o con clases UML estándar que implementen las otras capas de la aplicación utilizando de patrones arquitectónicos y de diseño. Por tanto, el uso de E-WAE y WAE4x para modelar la capa de presentación y UML para modelar las otras capas, permiten modelar una aplicación empresarial completa usando los patrones multicapa tales como (Alur et al., 2003; Fowler, 2002): *application services* para implementar los servicios de capa de negocio, *transfer objects* para mover datos entre capas, *data access objects* o *domain store* para la persistencia de objetos de negocio, *web service broker* para la publicación de servicios en una arquitectura orientada a servicios (SOA) (Erl, 2007) o *service activator* para la publicación de servicios en una arquitectura dirigida por eventos (EDA) (Etzion & Niblett, 2010). Estos patrones externos a la capa de presentación tienen que ser manualmente incluidos en los modelos y no se incluyen en los ejemplos, pero los elementos *Business Logic* son el punto de entrada hacia ellos.

Debido a que los perfiles WAE4x son meta-modelos PSM, su semántica se define en términos de los componentes implementados en los frameworks de desarrollo. Esta semántica facilita leer los modelos pensando en términos de los correspondientes componentes y además evita la ambigüedad en el modelado. Por tanto, las reglas de transformación definen una especificación ejecutable de los modelos clarificando su transformación a código y, en última instancia, cerrando la brecha entre modelos y código. Aunque estos modelos facilitan la transformación manual a código, el código presentado en este ejemplo ha sido obtenido automáticamente en Borland Together desde los modelos WAE4x mostrados en las Figuras 4.6 y 4.7 del capítulo anterior.

5.3.1.3 Generación de mockups

La Figura 5.4 muestra la estructura de los proyectos automáticamente generados en Eclipse desde el modelo WAE4JSF (a), y en Visual Studio desde el modelo WAE4.NET (b). Por simplicidad, la Figura 5.5 sólo muestra el código generado que define la página *ResourcePage* y los componentes que implementan la lógica navegacional en el caso de uso de eliminación de entidades *Recurso*. La página *ResourcePage* mostrada en la Figuras 5.5a y 5.5b contienen los elementos de presentación estándar y los componentes que lanzan el proceso de eliminación de recursos, y las Figuras 5.5c y 5.5d muestran los componentes que soportan el proceso navegacional, todos generados a partir de las relaciones navegacionales y sus atributos en los modelos WAE4x.

Por medio de la ejecución de las reglas de transformación automáticas, se genera el código sin intervención del usuario, incluyendo las páginas JSF, managed beans, el fichero *faces-config.xml*, clases Java (generadas desde las clases UML estándar incluidas en el modelo WAE4JSF), y la estructura de directorios siguiendo la disposición estándar que permite que las aplicaciones JSF sean desplegadas y ejecutadas en Eclipse. Por supuesto que las páginas JSF tienen que ser complementadas con el contenido que define la estructura de presentación y las clases pertenecientes a las capas de negocio e integración tienen que ser implementadas con el código de aplicación, pero el esquema de las páginas JSF, incluyendo formularios, componentes de entrada de datos estándar, botones, enlaces y managed beans se definen en el proyecto Java, permitiendo la ejecución directa del aspecto navegacional de la capa de presentación Web. Los componentes equivalentes son generados para las aplicaciones ASP.NET MVC. Las páginas son funcionales, de forma que, una vez cargadas y compiladas pueden ser usadas como mockups navegacionales, como se ilustra en la Figura 5.6 en Eclipse para JSF, y 5.7 en Visual Studio para ASP.NET MVC.

Figura 5.4: Proyectos generados automáticamente. (a) Proyecto generado automáticamente en Eclipse que implementa la eliminación de recursos en OdAJ2EE. (b) Proyecto generado automáticamente en Visual Studio que implementa la eliminación de recursos en OdAJ2EE

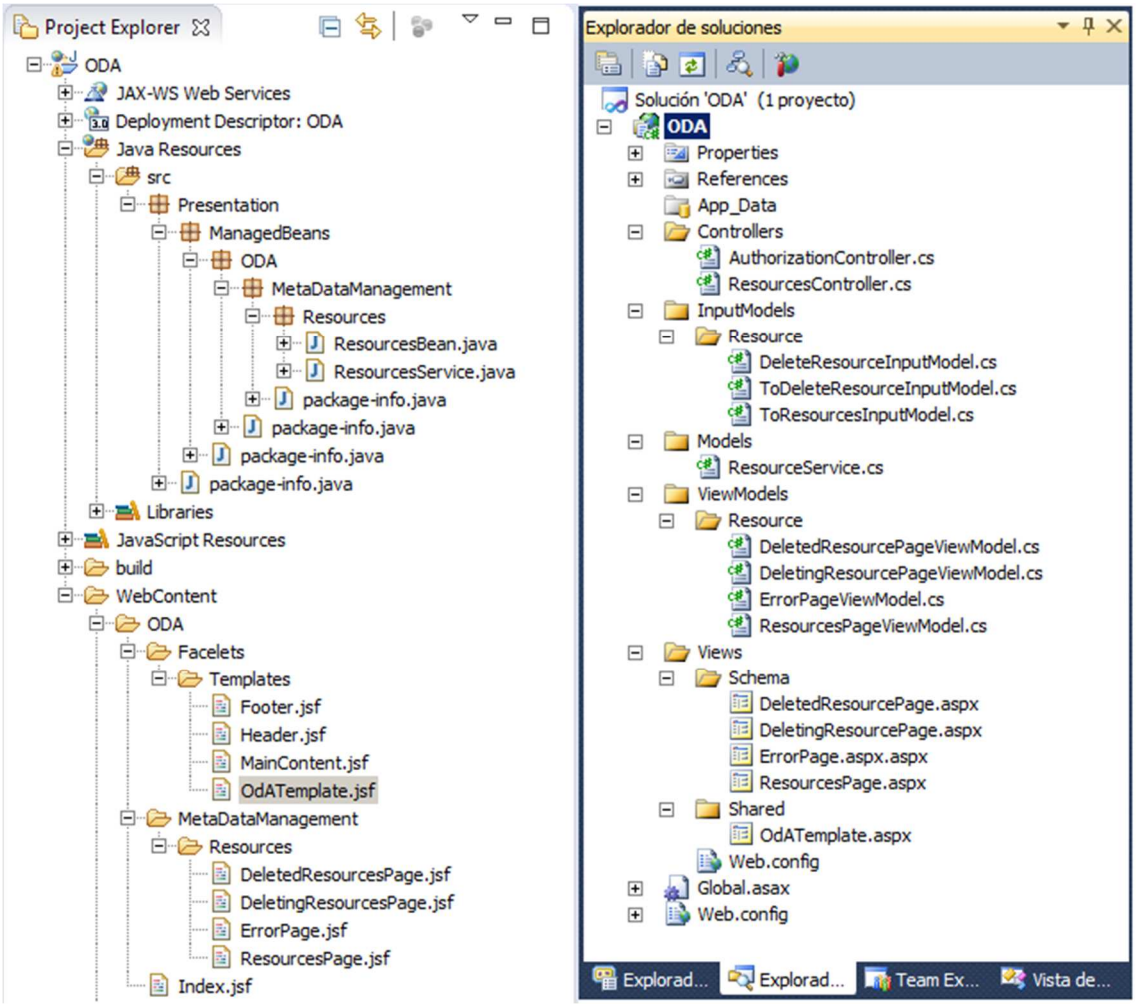


Figura 5.5: Código generado automáticamente. (a) Código generado automáticamente para la página *ResourcePage* en JSF. (b) Código generado automáticamente para la página *ResourcePage* en ASP.NET MVC. (c) Clase *Managed Bean* y reglas de navegación automáticamente generadas dentro del fichero *faces-config.xml* en JSF. (d) Clase *Controller* con sus acciones que soportan la navegación en ASP.NET MVC

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:c="http://java.sun.com/jsf/core"
      xmlns:ui = "http://java.sun.com/jsf/facelets"
      xmlns:h = "http://java.sun.com/jsf/html">
<head>
</head>
<h:body>
<ui:composition template="/ODA/Facelets/Templates/
                          OdATemplate.xhtml">
<ui:define name="MainContent">
  <h:form>
    <h:dataTable value="#{ResourceBean.resources}"
                 var="Resource">
      <h:column>
        #{Resource.id}
      </h:column>
      <h:column>
        #{Resource.description}
      </h:column>
      <h:column>
        <h:commandLink value="Eliminar"
           action="#{ResourceBean.
```

```
<%@          Page          Title=""          Language="C#"
MasterPageFile="~/Views/Shared/Site.Master"
Inherits="System.Web.Mvc.ViewPage<ODA.ViewModels.
ResourcesPageViewModel>" %>

<asp:Content                                  ID="Content2"
ContentPlaceHolderID="MainContent" runat="server">
 <table id="ResourcesTable">
   <tbody>
     <% foreach (ODA.ViewModels.Resource resource
                  in Model.Resources)
        { %>
      <tr>
        <td>
          <%=resource.Id %>
        </td>
        <td>
          <%=resource.Description %>
        </td>
        <td>
          <%: Html.ActionLink("Eliminar",
             "ToDeleteResource", resource.Id)%>
```

```
                    DeletingResource(Resource)}">
          <!-- TODO: Define the request
                    params as follows:
                <f:param name="paramName"
                         value="paramValue"/>
            -->
        </h:commandLink
      </h:column>
    </h:dataTable>
   </h:form>
</ui:define>
</ui:composition>
</h:body>
</html>
```

(a)

```
                //TODO : Define the parameters to
              //pass to action as follows
              //new { paramName = paramValue )
            </td>
          </tr>
          <%} %>
        </tbody>
    </table>
</asp:Content>
```

(b)

```
package ODA;

public class ResourceBean
{
  public String DeletingResource(Resource resource)
  {
    // TODO : Get the request parameters
    // Orchestrate the backend of the system
    // Return an outcome value between:
    // {"ToDeletingResource", }
    // According to boolean conditions:
    // { true,

    return "ToDeletingResource";
   }

   public String DeleteResource()
   {
     // TODO : Get the request parameters
     // Orchestrate the backend of the system
     // Return an outcome value between:
     // {"ToDeletedResource",
     //  "ToErrorResource"}
     // According to boolean conditions:
     // {Deleted == true, Deleted == false}

     return "ToDeletedResource";
     //return "ToErrorResource";
   }
}
<faces-config>
 <managed-bean>
   <managed-bean-name>resourceBean</managed-bean-name>
   <managed-bean-class>ODA.ResourceBean
   </managed-bean-class>
   <managed-bean-scope>session</managed-bean-scope>
 </managed-bean>
 <navigation-rule>
   <from-view-id>
     /ODA/Resources/ResourcesPage.xhtml
   </from-view-id>
   <navigation-case>
     <from-outcome>ToDeletingResource</from-outcome>
     <to-view-id>
       /ODA/Resources/DeletingResourcesPage.xhtml
     </to-view-id>
   </navigation-case>
 </navigation-rule>
 <navigation-rule>
   <from-view-id>
     /ODA/Resources/DeletingResourcesPage.xhtml
   </from-view-id>
   <navigation-case>
     <from-outcome>ToDeletedResource</from-outcome>
     <to-view-id>
       /ODA/Resources/DeletedResourcePage.xhtml
     </to-view-id>
   </navigation-case>
 </navigation-rule>
 <navigation-rule>
   <from-view-id>
     /ODA/Resources/DeletingResourcesPage.xhtml
   </from-view-id>
   <navigation-case>
     <from-outcome>ToErrorResource</from-outcome>
     <to-view-id>
       /ODA/Resources/ErrorPage.xhtml
     </to-view-id>
   </navigation-case>
  </navigation-rule>
</faces-config>
```

(c)

```
namespace ODA.Controllers
{
  public class ResourcesController : Controller
  {
   [HttpGet]
   [ActionName("ToResources")]
   [Authorize(Roles = "Admin")]
   public ActionResult ToResources(ToResourcesInputModel
                                   inputModel)
   {
     // Orchestrate the back end of the system
     // Get ViewModel data transfer object for the next
     // view
     // Navigate to the next view
     ResourcesPageViewModel resourcesPageViewModel =
     ResourceModel.GetResourcesPageViewModel();

     return                          View("ResourcesPage",
resourcesPageViewModel);
   }

   [HttpGet]
   [ActionName("ToDeleteResource")]
   [Authorize(Roles = "Admin")]
   public ActionResult
          ToDeleteResource(ToDeleteResourceInputModel
                           inputModel)
   {
     // Orchestrate the back end of the system
     // Get ViewModel data transfer object for the next
     // view
     // Navigate to the next view
     DeletingResourcePageViewModel
     deletingResourcePageViewModel =
     ResourceModel.GetDeletingResourcePageViewModel
     (inputModel);
     return View("DeletingResourcesPage",
                 deletingResourcePageViewModel);
   }

   [HttpGet]
   [ActionName("DeleteResource")]
   [Authorize(Roles = "Admin")]
   public ActionResult
          DeleteResource(DeleteResourceInputModel
                         inputModel)
   {
     // Orchestrate the back end of the system
     // Get ViewModel data transfer object for the next
     //view
     // Navigate to the next view
     DeletedResourcePageViewModel
     deleteResourcePageViewModel =
     ResourceModel.GetDeletedResourcePageViewModel();
     return View("DeletedResourcesPage",
                 deleteResourcePageViewModel);

     // ErrorPageViewModel errorPageViewModel =
     // ResourceModel.GetErrorPageViewModel();

     //return View("ErrorPage", errorPageViewModel);
   }
  }
}
```

(d)

De esta forma, E-WAE facilita del uso de mockups en el proceso de desarrollo como lo promueven las metodologías ágiles. Además, debido a que E-WAE usa notaciones UML y las transformaciones PIM a PSM y PSM a código no se ejecutan por medio de generadores propietario de código, los mockups pueden ser reutilizados completamente

en el proceso de desarrollo. Esta característica es posible gracias a que los modelos que soportan los mockups son modelos bien definidos que pueden ser manualmente extendidos siguiendo arquitecturas concretas. Cabe destacar que aunque las transformaciones PSM a código están definidas explícitamente, para hacer el enfoque usable, se requiere de su ejecución automática. Sin embargo, de ningún modo esto significa que el enfoque dependa de herramientas específicas. Modelos y código pueden evolucionar independientemente sin necesidad de herramientas propietarias que los sincronize, y con la posibilidad de que la generación automática de código pueda ser implementada libremente usando cualquier tecnología y/o herramienta.

Como puede observarse, las relaciones navegacionales que modelan la navegación estática generan enlaces completamente funcionales en los mockups proporcionando una experiencia real de navegación a través de la aplicación. Las relaciones navegacionales que modelan la navegación dinámica, teniendo en cuenta que los managed beans y controladores se generan automáticamente, obliga a que estos deban ser editados manualmente para proporcionar navegaciones que proporcionen acceso a las páginas destino en términos de valores de prueba que en última instancia elimina la necesidad de ejecución de lógica de negocio. Por ejemplo en las Figuras 5.5c y 5.5d, se puede retornar un valor que da acceso a la página de eliminación exitosa de recurso por el managed bean o controller respectivamente para simular la eliminación correcta sin ejecutar lógica de negocio. También pueden retornarse de manera aleatoria valores de éxito o fracaso. Por supuesto, esto es útil solo para propósito de prueba de navegación en el mockup, pero proporciona una visión real de la capa de presentación sin necesidad de implementar lógica de negocio y con la capacidad de reutilizar todos los elementos de la capa de presentación automáticamente generados en la aplicación final.

Figura 5.6: Mockup JSF automáticamente generado

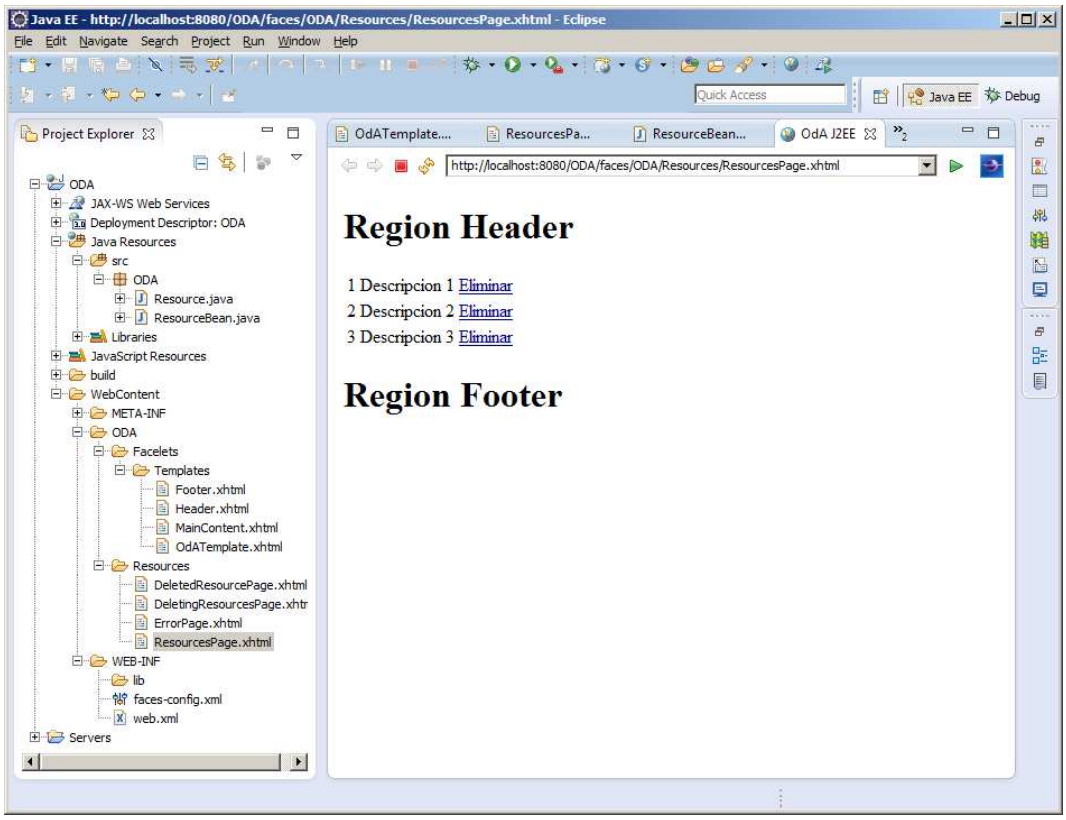
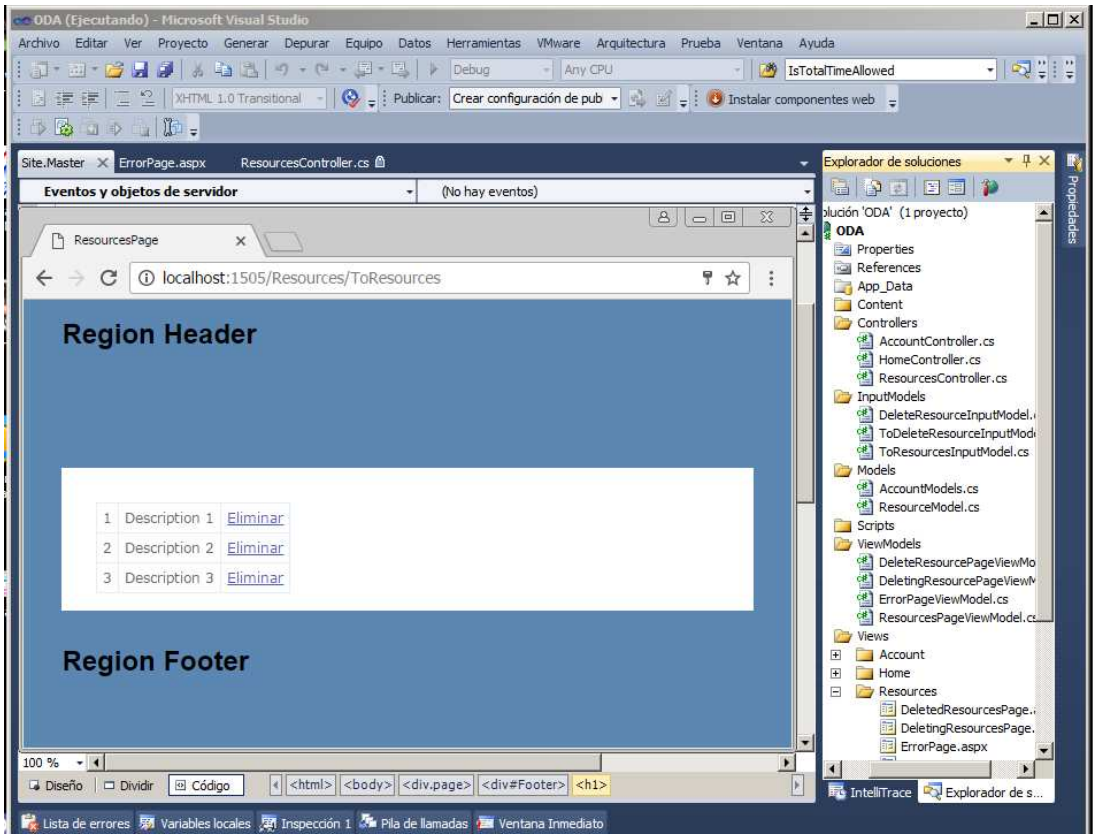


Figura 5.7: Mockup ASP.NET MVC automáticamente generado



5.3.2 Proceso de compra *on-line* en Amazon

El modelado de sistemas de compra electrónica es uno de los ejemplos más comunes encontrados en la literatura. Por tanto, este trabajo provee mapas navegacionales E-WAE para caracterizar la capa de presentación Web de estos sistemas. En concreto, en esta sección se utiliza E-WAE para modelar el proceso de compra *on-line* implementado por la aplicación Web de Amazon desde el punto de vista de la capa de presentación. Dado que E-WAE es una notación basada en UML, promueve el uso de diagramas de casos de uso UML y diagramas de actividad UML para modelar los requisitos, así como diagramas de despliegue UML para modelar la arquitectura hardware y software.

El proceso de compra *on-line* en Amazon está compuesto por varias etapas secuenciales que incluyen autenticación, selección de dirección de envío, de opciones de envío, de cantidades, de formas de pago y resumen del pedido. Como se ha comentado, en E-WAE los datos usados por la capa de presentación se modelan como clases estereotipadas con *DataCarrier* y los comandos y sus funciones, que representan ejecución de lógica de aplicación, se modelan como clases estereotipadas con *Command* y *Function* respectivamente.

Aunque cada etapa se modela por medio de un mapa navegacional independiente, clases en un mapa referencian a clases de otros mapas modelando todas las transiciones posibles en el proceso. Esta característica es usual en el modelado UML, donde las clases pueden ser definidas en paquetes concretos, pero pueden referenciar a otras clases de otros paquetes. Nótese que en las figuras, con el fin de facilitar la lectura, los atributos de los estereotipos definidos en E-WAE (visibles sólo en las vistas de propiedades de las herramientas CASE UML) se muestran por medio de notas UML en los diagramas.

5.3.2.1 Modelo E-WAE para las etapas de autenticación y selección de dirección de envío

La Figura 5.8 muestra una captura de pantalla de estas dos etapas del proceso. A la izquierda se muestra la pantalla que permite a los usuarios autenticarse en el sistema. Si el proceso de autenticación tiene éxito, la aplicación navega a la pantalla de la derecha, que permite a los usuarios seleccionar la dirección de envío.

Figura 5.8: Autenticación y selección de dirección en el proceso de compra *on-line* de Amazon

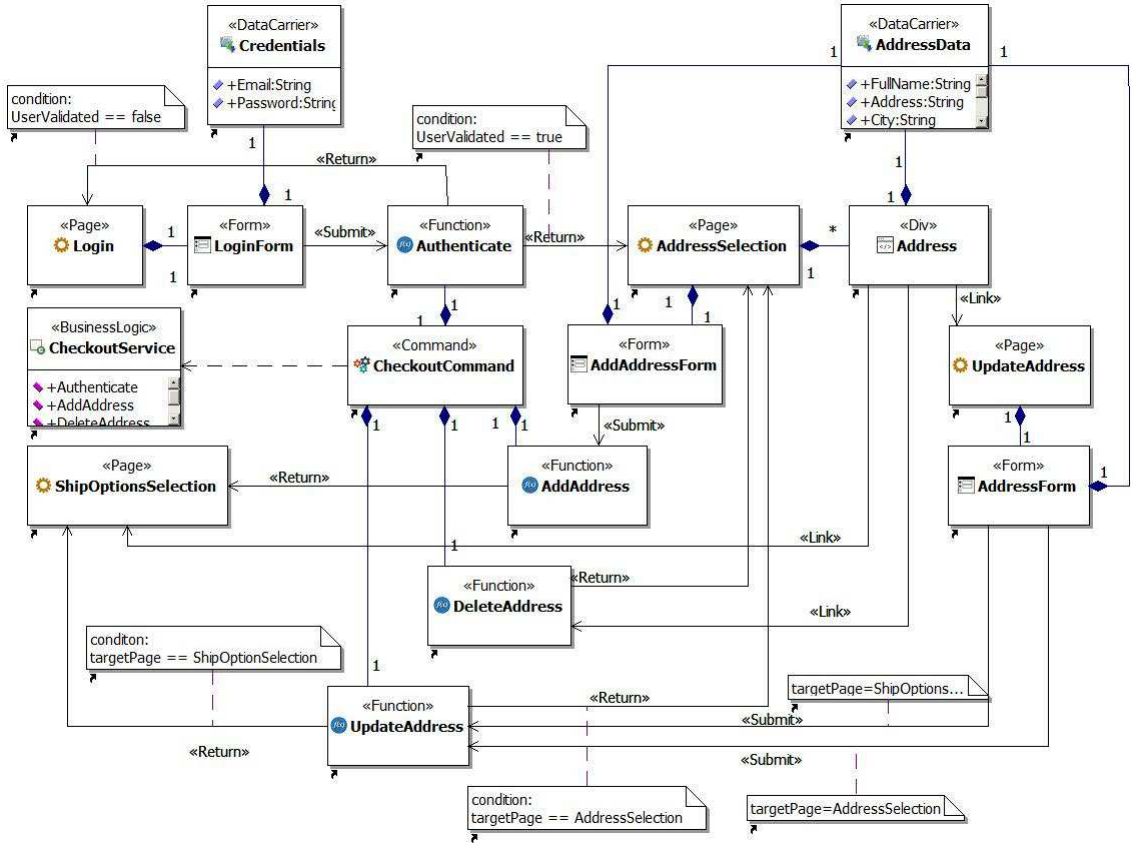
La Figura 5.9 muestra el diagrama E-WAE que modela las funcionalidades soportadas por estas pantallas. Como muestra el modelo, el proceso de autenticación es ejecutado a través de la página *Login*, la cual contiene un formulario, compuesto por los datos *Credentials*, que tiene la capacidad de lanzar un proceso navegacional enviando los datos del formulario a la función *Authenticate*, la cual representa el punto de acceso del proceso computacional responsable de implementar el mecanismo de autenticación. La página destino depende de la condición booleana alcanzada por la ejecución del proceso computacional, que puede ser la página *AddressSelection* si el proceso de autenticación es tiene éxito, o la página *Login* en caso contrario.

Siguiendo el mapa navegacional E-WAE de la Figura 5.9, la página *AddressSelection* contiene el formulario *AddAddressForm* para componer y crear una nueva dirección representada por *AddressData* y navegar a la página de selección de opciones de envío por medio de la función *AddAddress*. Además, la página *AddressSelection* está compuesta por divisiones *Address*, las cuales representan secciones en la página que contienen, cada una, una dirección previamente definida por el usuario actual. Desde cada una de estas divisiones, los usuarios pueden navegar a la página de selección de opciones de envío, indicando que los usuarios eligen la dirección seleccionada como dirección de envío; eliminar la dirección seleccionada por medio de la función *DeleteAddress*, permaneciendo en la misma página; o actualizar la dirección seleccionada navegando a la página *UpdateAddress*. Esta página está compuesta por un formulario para recolectar los datos a actualizar y navegar a la página *AddressSelection* o a la página

ShipOptionsSelection través de la función *UpdateAddress* dependiendo de los parámetros definidos en el valor etiquetado *Parameters* del estereotipo *Submit*.

Nótese que la conexión entre la capa de presentación y el resto de capas se representa por la dependencia desde el elemento *Command* al elemento *Business Logic*. Este último elemento caracteriza interfaces de software que pueden ser localmente implementadas por servicios de aplicación, delegados de negocio que localizan servicios remotos, o cualquier otro patrón multicapa que implemente la lógica de negocio relacionado con procesos de compra on-line.

Figura 5.9: Autenticación y selección de dirección en el proceso de compra *on-line* de Amazon con E-WAE

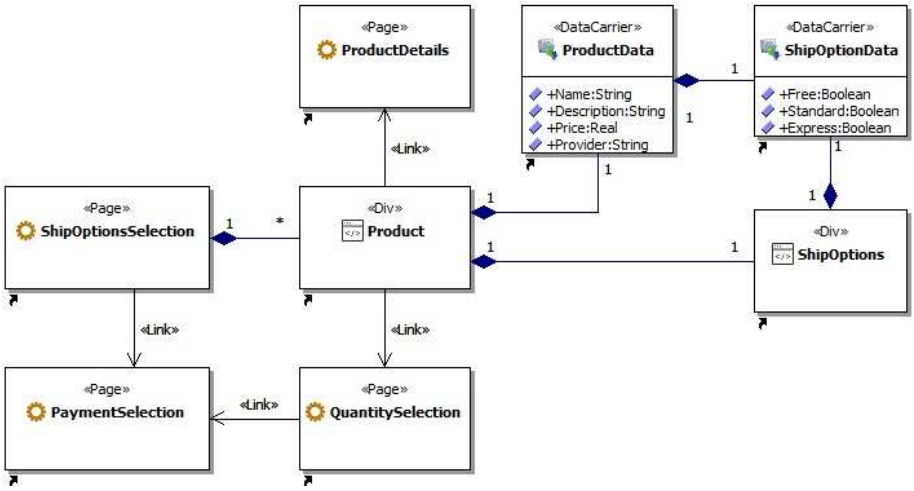


5.3.2.2 Modelo E-WAE para la etapa de selección de opciones de envío

La Figura 5.10 muestra el diagrama E-WAE que modela la etapa de selección de opciones de envío. La página *ShipOptionsSelection* muestra todos los productos a comprar en diferentes secciones representadas por las clases *Product*. Cada una de estas secciones permite a los usuarios navegar a la página *ProductDetails*, la cual muestra la información detallada de cada producto; contiene secciones *ShipOptions* para mostrar y permitir seleccionar las opciones de envío de cada producto; y permite navegar a la página

QuantitySelection. Nótese que cada sección *Product* está soportada por los respectivos elementos *DataCarrier* para mostrar y guardar los datos modificados en la interfaz de usuario, y que desde la página *ShipOptionsSelection* es posible navegar directamente a la página *PaymentSelection* saltándose la etapa de selección de cantidades.

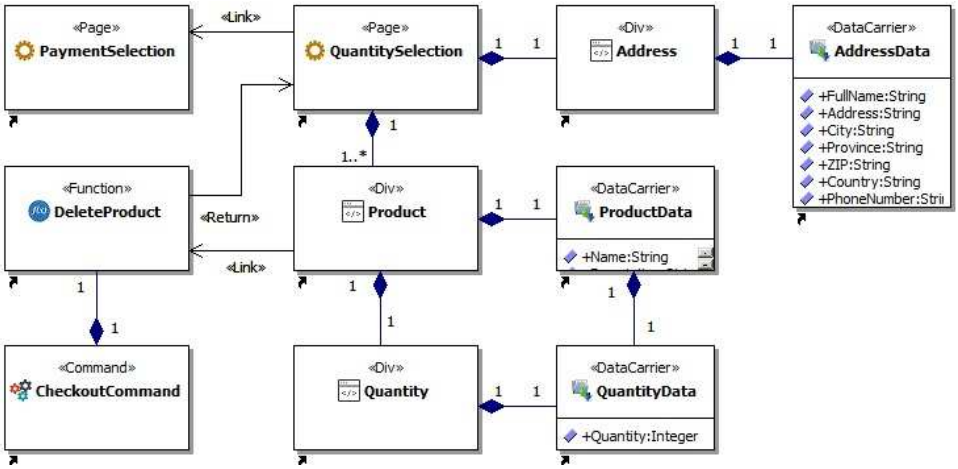
Figura 5.10: Selección de opciones de envío en el proceso de compra *on-line* de Amazon con E-WAE



5.3.2.3 Modelo E-WAE para la etapa de selección de cantidades

La Figura 5.11 muestra el diagrama E-WAE que modela la etapa de selección de cantidades. La página *QuantitySelection* muestra cada uno de los productos a comprar en una sección *Product* diferente, cada una de las cuales, a su vez, se compone de una sección *Quantity* que permite editar las cantidades de los productos a comprar. Además, en esta etapa, desde cada sección *Product* es posible eliminar el producto seleccionado, permaneciendo en la misma página, se muestra la dirección de envío seleccionada y es posible navegar a la página *PaymentSelection*.

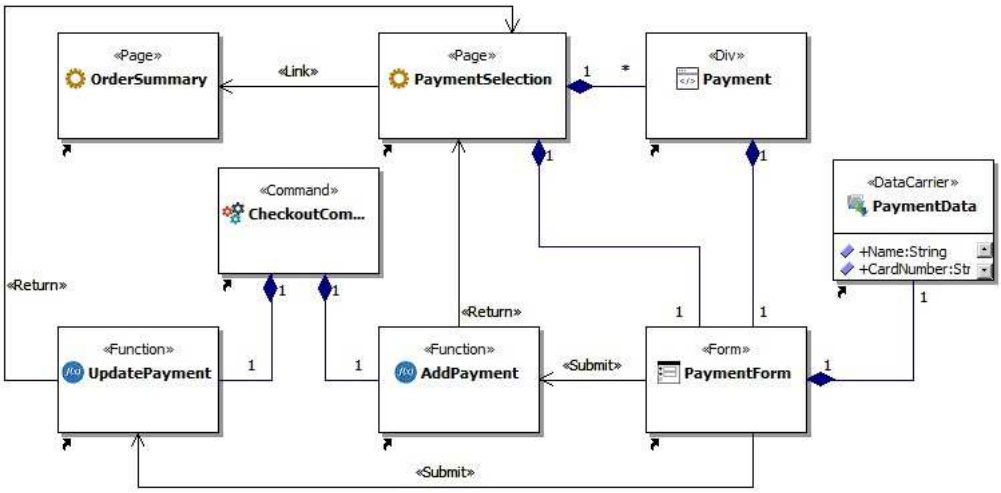
Figura 5.11: Selección de cantidades en el proceso de compra *on-line* de Amazon con E-WAE



5.3.2.4 Modelo E-WAE para la etapa de selección de formas de pago

La Figura 5.12 muestra el diagrama E-WAE que modela la etapa de selección de formas de pago. A través de las secciones *Payment*, la página *PaymentSelection* muestra todos los medios de pago previamente definidos por el usuario actual, permitiendo además actualizarlos o crear nuevos. Los usuarios pueden seleccionar uno de los medios de pago y navegar a la página *OrderSummary*.

Figura 5.12: Selección de formas de pago en el proceso de compra *on-line* de Amazon con E-WAE

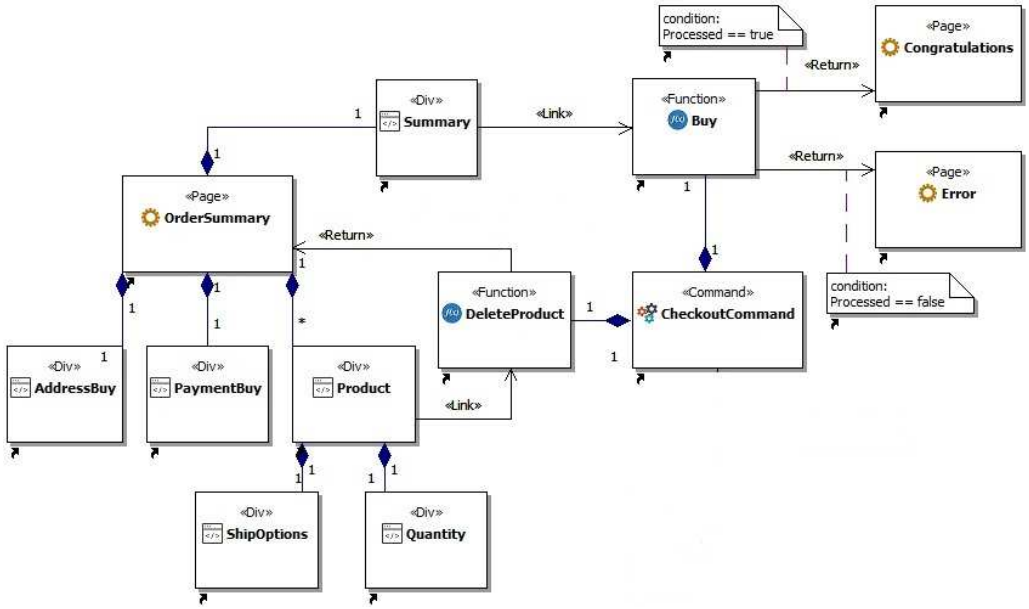


5.3.2.5 Modelo E-WAE para la etapa de resumen del pedido

La Figura 5.13 muestra el diagrama E-WAE que modela la etapa de resumen del pedido. La página *OrderSummary* muestra todos los datos seleccionados por el usuario en el proceso de compra. Como se puede ver, la página presenta la dirección de envio del pedido, su forma de pago y cada producto que lo compone se presenta en su propia

división y con sus propias características, permitiendo además ser eliminado en esta última etapa. La página *OrderSummary* permite lanzar el último proceso navegacional enviando todos los datos del pedido a la función *Buy*, la cual representa el proceso computacional responsable de implementar el proceso de compra. La página destino depende de la condición booleana alcanzada por la función, retornando el página *Congratulations*, si el proceso de compra es exitoso o la página *Error* en caso contrario.

Figura 5.13: Resumen del pedido en el proceso de compra *on-line* de Amazon con E-WAE



5.3.2.6 Modelos WAE4x para las etapas de autenticación y selección de dirección de envío

La Tabla 5.4 define las reglas para transformar modelos E-WAE en modelos WAE4x. Para facilitar la ilustración, en esta sección sólo se genera un diagrama simple para cada plataforma. Si se aplican las reglas de transformación de E-WAE a WAE4x al diagrama mostrado en la Figura 5.9, se obtienen los diagramas mostrados en las Figuras 5.14 y 5.15. Cabe destacar que, aunque las reglas de transformación pueden aplicarse manualmente, estos diagramas han sido obtenidos automáticamente en la herramienta CASE Borland Together aplicando las transformaciones QVT *Operational Mappings* que implementan las reglas definidas. Los modelos WAE4x obtenidos, son modelos PSM con una semántica dada por los conceptos y componentes implementados en los frameworks de desarrollo, por lo cual estos modelos evitan la ambigüedad del modelado cerrando la brecha entre modelos y código, y finalmente facilitando su transformación a código ejecutable.

Figura 5.14: Autenticación y selección de dirección en el en el proceso de compra on-line de Amazon con WAE4JSF

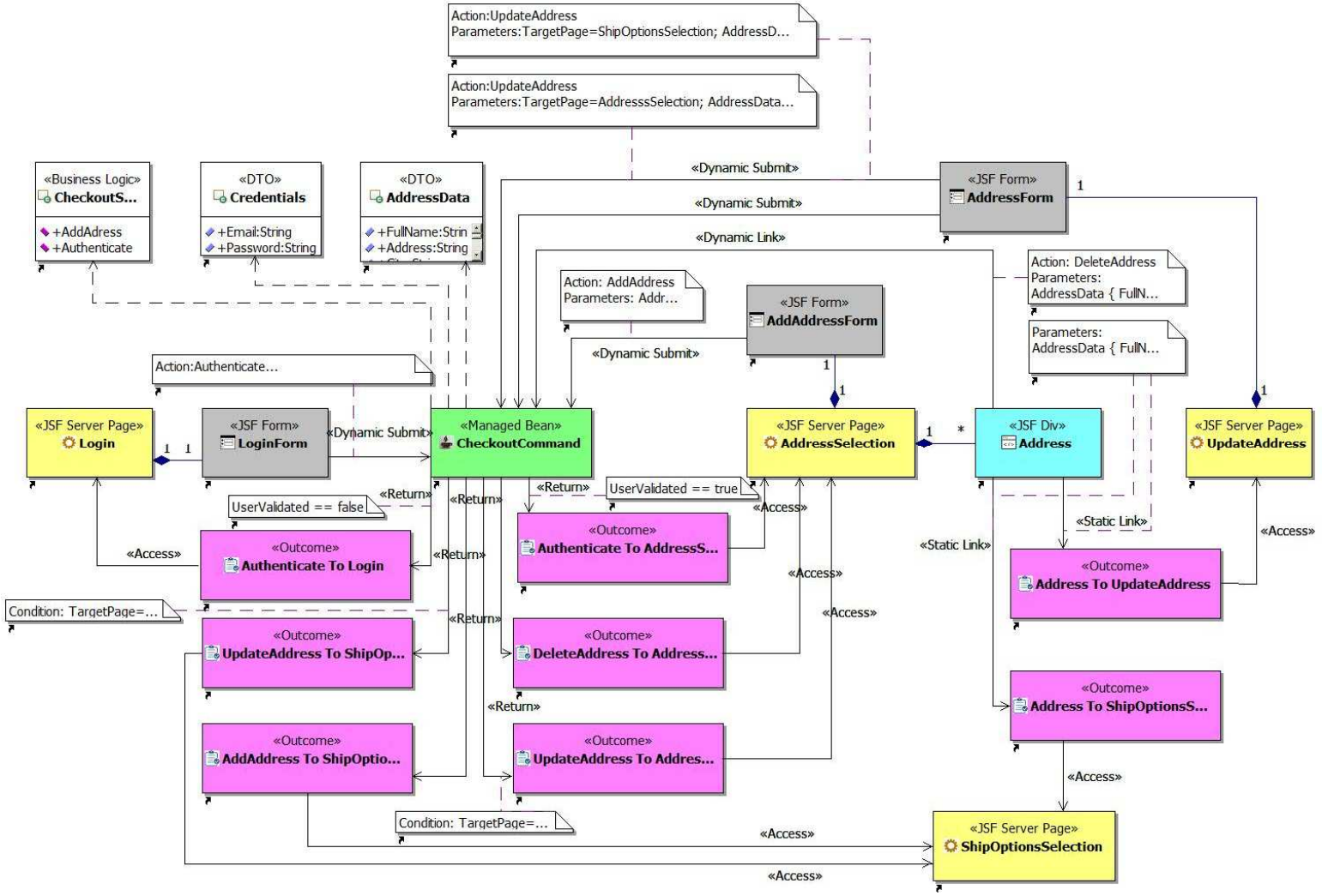
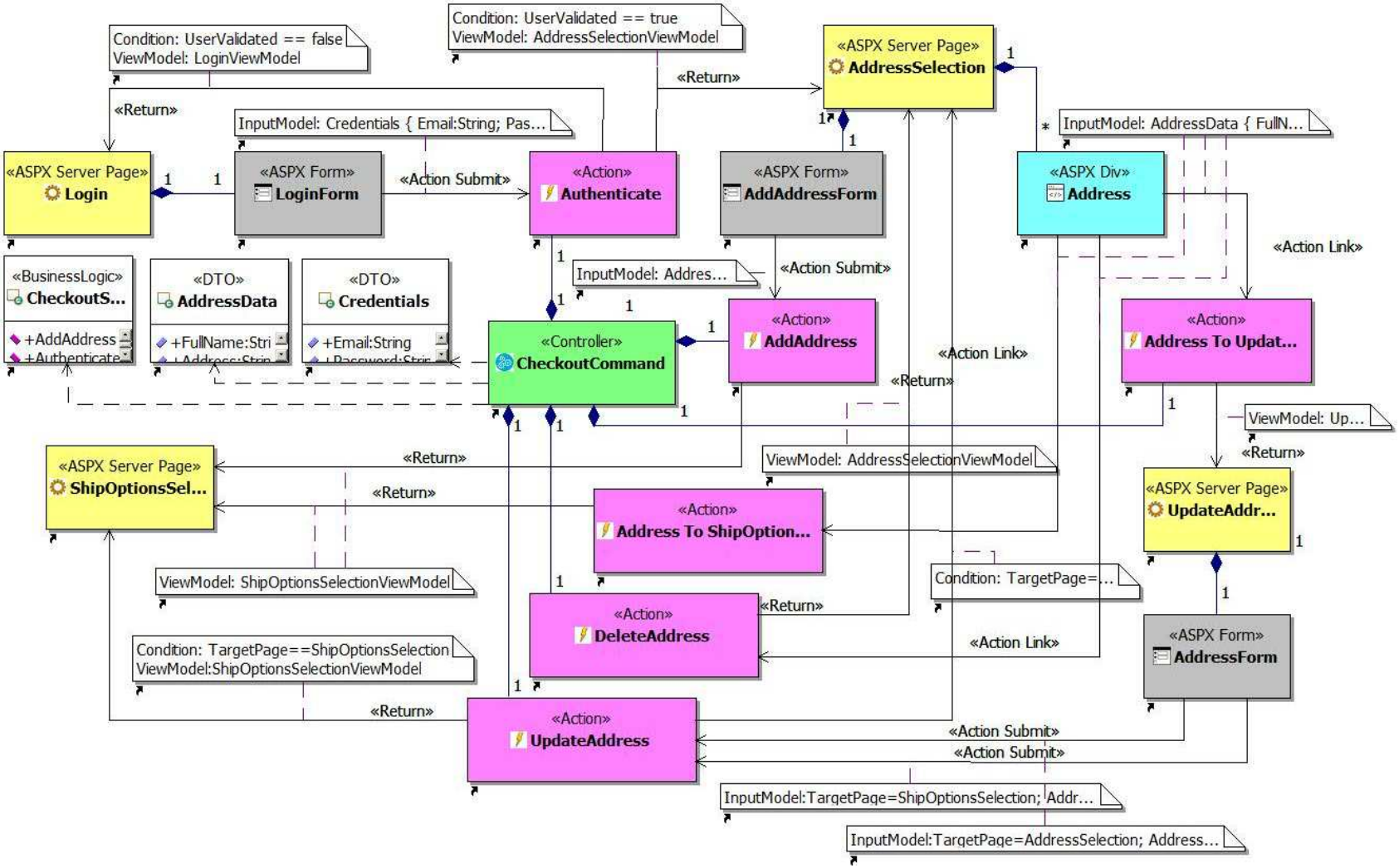


Figura 5.15 Autenticación y selección de dirección en el en el proceso de compra on-line de Amazon con WAE4.NET



5.3.2.7 Generación de mockups

En aras de la concisión, la Figura 5.16 sólo muestra el código generado automáticamente, desde los modelos WAE4x ilustrados en las Figuras 5.14 y 5.15, que define la página *Login* y los artefactos que implementan la lógica navegacional involucrada en el proceso de autenticación. La página *Login* definida en las Figuras 5.16a y 5.16b contiene los elementos de entrada estándar y el comando que lanza el proceso de autenticación. Las Figuras 5.16c y 5.16d muestran los componentes que soportan el proceso navegacional, generados todos a partir de las relaciones navegacionales y sus atributos en los modelos WAE4x. Las páginas son completamente funcionales, por tanto una vez cargadas y compiladas pueden ser usadas como mockups navegacionales, como muestra la Figura 5.17 para la aplicación JSF en Eclipse y la Figura 5.18 para la aplicación ASP.NET MVC en Visual Studio

Figura 5.16: Código automáticamente generado. (a) Código generado automáticamente para la página Login en JSF. (b) Código generado automáticamente para la página Login en ASP.NET MVC. (c) Clase Managed Bean y reglas navegacionales en el fichero faces-config.xml generados automáticamente en JSF. (d) Clase Controller y sus acciones generados automáticamente en ASP.NET MVC

```
<html

  xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:f="http://java.sun.com/jsf/core">
  <f:view xmlns:f="http://java.sun.com/jsf/core">
     <h:body>
        <h:form>
           <h:inputText
             id="Email"
             value="#{CheckoutCommand.Email}"/>
           <h:inputText
             id="Password"
             value="#{CheckoutCommand.Password}"/>
           <h:commandButton
             value="LoginFormToCheckoutCommand"
             action="#{CheckoutCommand.Authenticate}">

              <!-- TODO: Define the request
                   params as follows:
                   <f:param name="paramName"
                            value="paramValue"/>
              -->
           </h:commandButton>
         </h:form>
    </h:body>
  </f:view>
</html>
```

(a)

```
<%@ Page Title="LoginPage"
         Language="C#"
         Inherits="System.Web.Mvc.ViewPage
                   <Amazon.ViewModels.LoginViewModel>" %>

    <%: Html.BeginForm("LoginFormToCheckoutCommand",
                       "Authenticate",
                       "CheckoutCommand")
    {
        Html.TextBox("Email");
        Html.TextBox("Password");
        Html.Button("LoginFormToCheckoutCommand")

        //TODO : Define the parameters to
        //pass to action as follows
        //new { paramName = paramValue )
    }

    %>
</asp:Content>
```

(b)

```
package Presentation.ManagedBeans.Amazon.Checkout;

public class CheckoutCommand {

   private String Email;
   public String getEmail() {
             return Email;
   }
   public void setEmail (String newValue) {
     Email = newValue;
   }

   private String Password;
   public String getPassword() {
     return Password;
   }
   public void setPassword(String newValue) {
     Password = newValue;
   }

   public String Authenticate() {
```

```
public class SchemaController : Controller
{
  [HttpGet]
  [ActionName("Authenticate")]
  public ActionResult Authenticate(credentialsInputModel
  inputModel)
  {
   // Orchestrate the back end of the system
   // Navigate to one of the next views
   // according to boolean conditions:
   // {userValidated == true, userValidated == false}
   return RedirectToAction(
         "AddressSelectionViaGet",
         new { inputModel });

   return RedirectToAction(
         "LoginViaGet",
          new { inputModel });
  }

  [HttpGet]
```

```
   // TODO : Get the request parameters
   // Orchestrate the backend of the system
   // Return an outcome value between:
   // {"AuthenticateToAddressSelection",
       "AuthenticateToLogin"}
   // According to boolean conditions:
   // {userValidated == true, userValidated == false}

      return "Outcome";
   }
}
```

(c)

```
<faces-config>
 <managed-bean>
  <managed-bean-name>CheckoutCommand
  </managed-bean-name>
  <managed-bean-class>
     Presentation.ManagedBeans.Amazon.Checkout.
     CheckoutCommand
  </managed-bean-class>
<managed-bean-scope>session
  </managed-bean-scope>
 </managed-bean>
   <navigation-rule>
     <from-view-id>
        /Amazon/Checkout/Login.jsf
     </from-view-id>
     <navigation-case>
        <from-outcome>
           AuthenticateToAddressSelection
        </from-outcome>
        <to-view-id>
           /Amazon/Checkout/AddressSelection.jsf
        </to-view-id>
        <redirect />
      </navigation-case>
      <navigation-case>
        <from-outcome>
          AuthenticateToLogin
        </from-outcome>
        <to-view-id>
           /Amazon/Checkout/Login.jsf
        </to-view-id>
        <redirect />
      </navigation-case>
    </navigation-rule>
</faces-config>
```

(c)

```
  [ActionName("AddressSelectionViaGet")]
  public ActionResult AddressSelectionViaGet(object
  inputModel)
  {
    // Orchestrate the back end of the system
    // Get ViewModel transfer object for the next view
    // Navigate to the next view
    AddressSelectionViewModel AddressSelectionViewModel =
    Model.GetAddressSelectionViewModel();

    return View("AddressSelection",
                AddressSelectionViewModel);
  }

  [HttpGet]
  [ActionName("LoginViaGet")]
  public ActionResult LoginViaGet(object inputModel)
  {
    // Orchestrate the back end of the system
    // Get ViewModel transfer object for the next view
    // Navigate to the next view
    LoginViewModel LoginViewModel =
    Model.GetLoginViewModel();

    return View("Login",
                LoginViewModel);
  }
}
```

(d)

Figura 5.17: Mockup JSF automáticamente generado

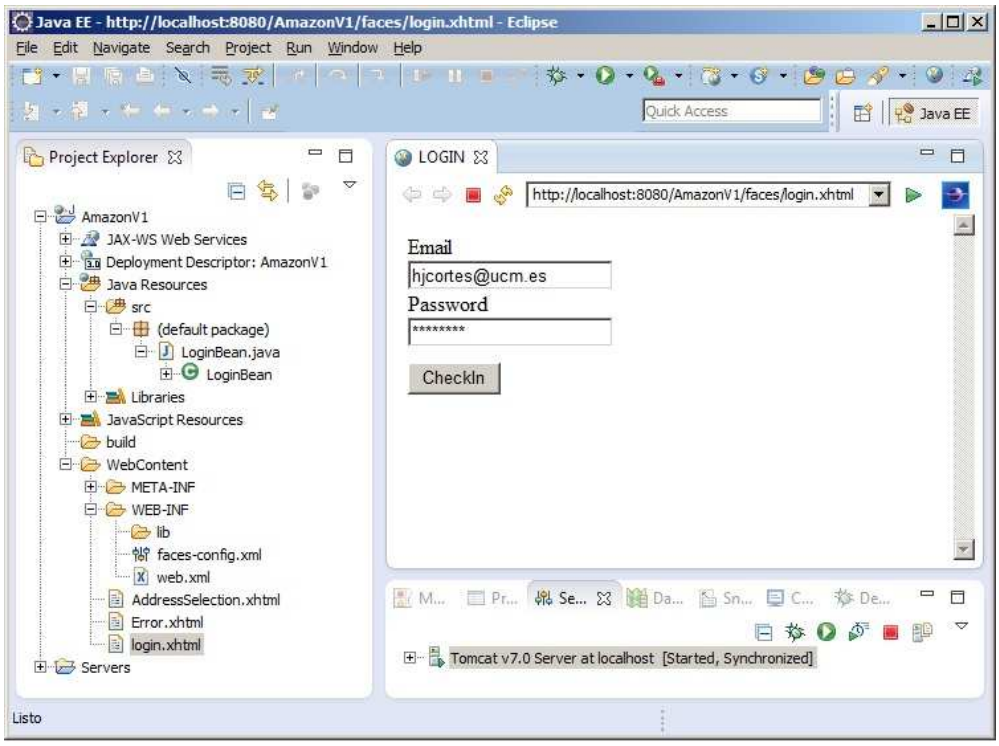
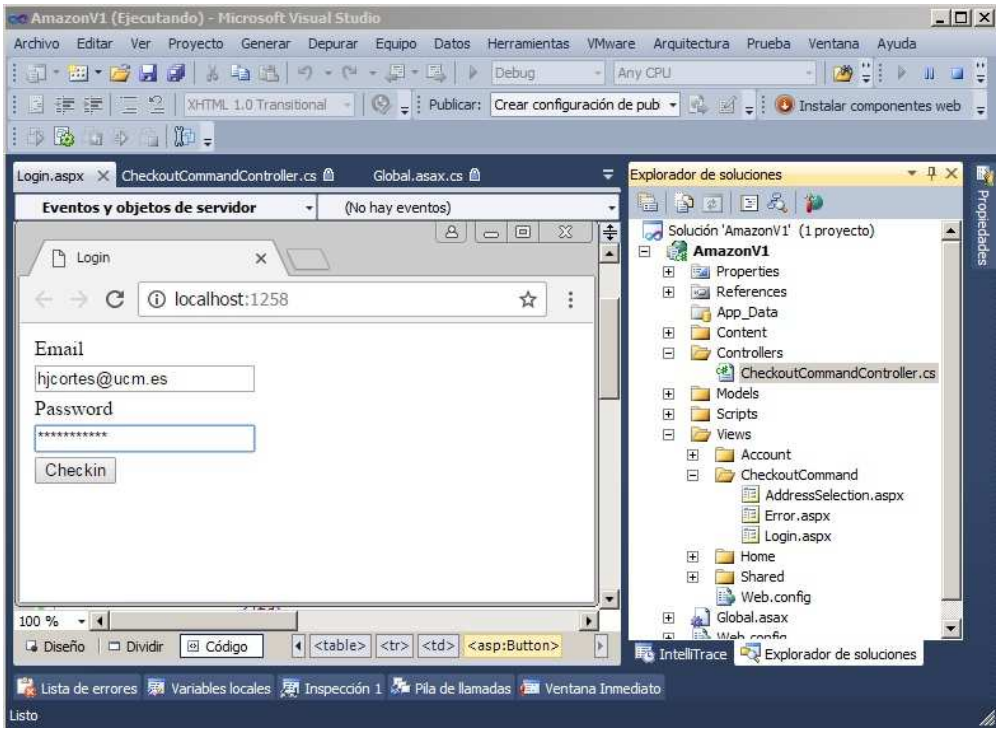


Figura 5.18: Mockup ASP.NET MVC automáticamente generado



5.3.3 Función de búsqueda en la Librería On-line del Congreso de USA

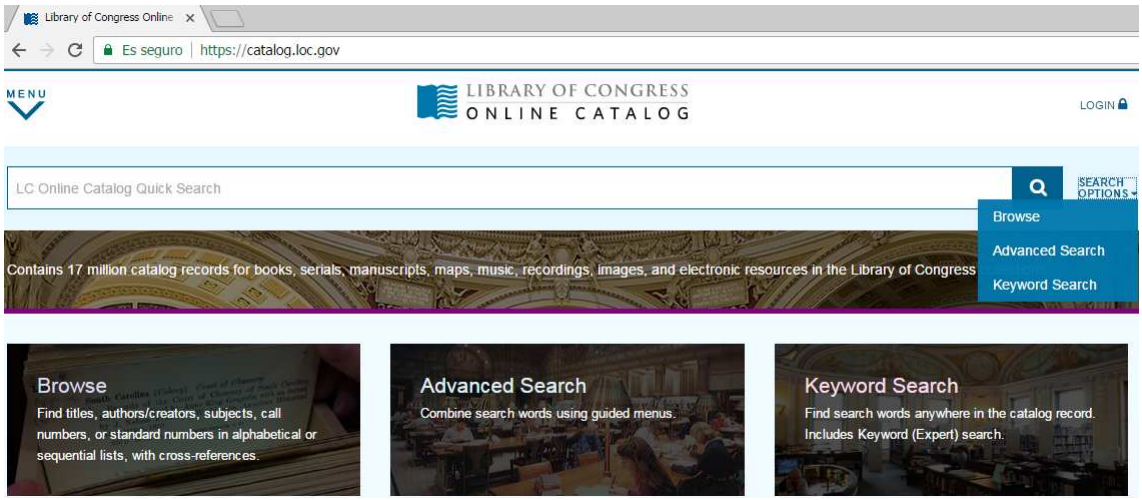
La Librería On-line del Congreso de USA[11] permite el acceso a aproximadamente 17 millones de registros que representan libros, ficheros digitales, manuscritos, registros de sonido y material visual, entre otros. El conjunto de registros se almacena en una única base de datos, por lo que la aplicación Web ofrece potentes ayudas de búsqueda a los usuarios. El objetivo de este ejemplo es mostrar la factibilidad de usar E-WAE para modelar menús navegacionales, aunque estos no se modelan desde la perspectiva de la presentación sino desde la perspectiva navegacional, por tanto, los enlaces que definen los menús son incluidos en las páginas pero no son específicamente formateados como menús.

La Figura 5.19 muestra una captura de pantalla de la librería. La estructura está basada en una plantilla con una cabecera común, la cual contiene un menú principal, un logo, un enlace a una página de *Login*, un formulario que permite ejecutar una búsqueda rápida y un menú simple con tres ítems los cuales cambian de acuerdo a la página presentada. Por simplicidad solo este simple menú y el menú que lo sobrescribe de acuerdo a la página presentada se modela en este ejemplo. En la figura, la página mostrada es la página *Index*,

[11] Accesible en https://catalog.loc.gov/

que da acceso a las mismas páginas que el menú simple: *Browse Page, Advanced Search Page y Keyword Search Page.*

Figura 5.19: Captura de pantalla de la Librería on-line del Congreso de USA. Nótese el menú navegacional a la derecha de la captura, bajo la imagen de la lupa

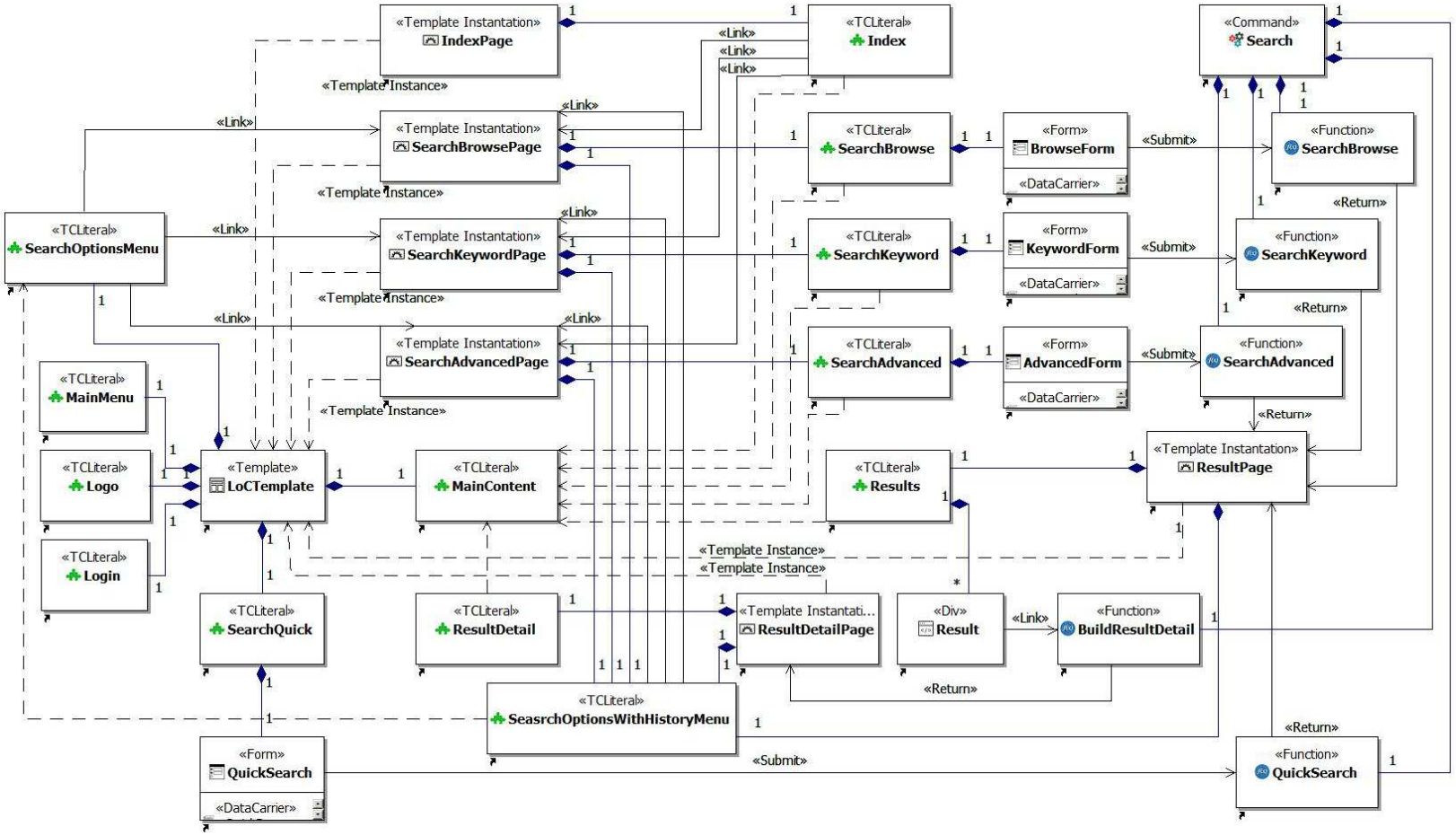


La Figura 5.20 caracteriza la librería en términos de un modelo UML de clases extendido con el perfil E-WAE. Como muestra la figura, todas las páginas instancian la misma plantilla, compartiendo las regiones definidas en ella. El menú simple, representado por el elemento *SearchOptionsMenu,* define tres enlaces que dan acceso a las páginas *SearchBrowsePage, SearchKeywordPage y SearchAdvancedPage*. Nótese que estas páginas también pueden alcanzarse desde la página *IndexPage*, y que todas las páginas definen una región que sobrescribe la región *MainContent* definida en la plantilla *LocTemplate*, proporcionando cada página diferentes ayudas de búsqueda a los usuarios. Además, las páginas *SearchBrowsePage, SearchKeywordPage, SearchAdvancedPage, ResultPage* y *ResultDetailPage* definen el menu *SearchOptionsWithHistoryMenu*, el cual sobrescribe el menú simple *SearchOptionsMenu*. Por tanto, estas páginas muestran un menú extendido que contiene los mismos enlaces que el menú simple así como otros que no se incluyen en el diagrama en aras de la sencillez.

Todas las páginas tienen el mismo objetivo, llamar a la función de búsqueda para encontrar y presentar los registros que coincidan con los criterios de búsqueda seleccionados por los usuarios. Nótese que el contenido principal de cada página se compone de un formulario con un elemento *DataCarrier* que representa los criterios de búsqueda a aplicar en la función de búsqueda respectiva. Todas las funciones de búsqueda retornan la misma página *ResultPage* que presenta en secciones los registros encontrados

por medio de los elementos *Result*, que a su vez, permiten navegar a la página *ResultDetailPage* para mostrar los detalles de un registro concreto.

Figura 5.20: Búsqueda de la Librería on-line del Congreso de USA con E-WAE



6 CONCLUSIONES Y TRABAJO FUTURO

6.1 Conclusiones

El diseño, desarrollo y mantenimiento de aplicaciones Web es una labor compleja. Una prueba de ello es el intenso trabajo realizado por la comunidad de Ingeniería Web que ha producido un gran número de notaciones de diseño (enfoques orientados al modelado) y el desarrollo de herramientas comerciales diseñadas especialmente para el diseño y construcción de aplicaciones Web (enfoques orientados al código).

Aunque algunas de estas soluciones han tenido éxito en el contexto de las aplicaciones Web dirigidas por datos (*data-intensive Web applications*), todas ellas fuerzan prácticas específicas para diseñar y desarrollar las aplicaciones. Los enfoques orientados al modelado cambian el estilo de diseño basado en diagramas UML de clases y de secuencia, y el desarrollo basado en la arquitectura multicapa a su propio estilo, definido en cada enfoque. Además, por lo general, estos enfoques también definen su propia notación de diseño. Por otro lado, las herramientas comerciales, proveen algún tipo de notación visual para el desarrollo del código sin proporcionar una infraestructura de ingeniería de software dedicada a su diseño y mantenimiento. En ambos casos es muy difícil, si no imposible, incluir explícitamente patrones arquitectónicos y de diseño en los modelos, y requieren herramientas propietarias para generar código a partir de notaciones que son casi imposible de traducir directamente a código, por lo que el mantenimiento de éste se ve vinculado a la herramienta que lo generó.

Si se admite que la filosofía de modelado UML es la más ampliamente usada en la industria, que la arquitectura multicapa es la más común en el desarrollo de aplicaciones empresariales modernas, y que este tipo de aplicaciones usa frameworks empresariales para implementar su capa de presentación Web, actualmente los desarrolladores no cuentan con un enfoque que les permita modelar esta capa de presentación en términos de diagramas UML de clases y de secuencia.

Por tanto, uno de los objetivos de este trabajo es proporcionar a los desarrolladores un conjunto de extensiones UML que les permita generar modelos de la capa de presentación Web de aplicaciones empresariales modernas directamente traducibles a código, usando herramientas UML CASE genéricas, sin necesidad de aprender una nueva notación de diseño, sin la limitación de un conjunto predefinido de primitivas de modelado, sin necesidad de generadores de código que condicionen su mantenimiento y con todo el poder que la arquitectura multicapa proporciona.

Por otro lado, los enfoques orientados al modelado han demostrado que el uso de modelos PIM es muy valioso en el proceso de desarrollo. Los modelos PIM permiten expresar los conceptos del dominio independientemente de los conceptos de implementación. Por tanto, otro de los objetivos del presente trabajo es proveer a los desarrolladores con una notación de alto nivel que permita el desarrollo de modelos PIM para caracterizar la capa de presentación Web sin prescindir de las características anteriormente mencionadas.

El primer paso para cumplir con estos objetivos ha sido desarrollar NMMp, una notación que añade un nivel de abstracción sobre UML-WAE. NMMp permite representar mapas navegacionales en *estado puro*, es decir, independientes de detalles arquitectónicos y de conceptos de programación, permitiendo generar modelos sencillos que muestran una visión general de la aplicación facilitando la comunicación entre usuarios finales y desarrolladores. Además, NMMp incluye reglas de transformación para convertir de forma directa modelos NMMp en modelos UML-WAE que caracterizan diseños arquitectónicos detallados sin acoplarse a ninguna notación ni herramienta específica.

El enfoque NMMp considera los diagramas de diseño como representaciones abstractas de código y no como simples representaciones visuales de código (Booch et al., 2007; Rumbaugh et al., 2010). Por tanto, aunque con NMMp el código puede ser generado, no así la aplicación lista para ser ejecutada. Para salvar este obstáculo, se ha extendido UML-WAE para dotarlo con la capacidad de caracterizar modelos directamente traducibles a código ejecutable sin necesidad de herramientas específicas a las que vincular su

mantenimiento. Siguiendo la filosofía de UML-WAE se han caracterizado los componentes de dos frameworks empresariales, JSF y ASP.NET MVC usando perfiles UML. Estos perfiles, llamados WAE4JSF y WAE4.NET (abreviado WAE4x), permiten generar modelos PSM para caracterizar la capa de presentación Web de aplicaciones empresariales modernas.

Los perfiles WAE4x permiten a los desarrolladores modelar la capa de presentación Web utilizando conceptos y componentes implementados en frameworks específicos, y por tanto, modelar directamente código de capa de presentación, como UML hace en el resto de capas de una aplicación. Estos perfiles, como extensiones de UML-WAE, heredan todos sus beneficios, incluyendo la capacidad de integrarse con otros modelos UML que representen las otras capas de una aplicación a través de patrones arquitectónicos y de diseño.

Para probar la aplicabilidad de los perfiles WAE4x, se decidió utilizarlos en el desarrollo de la aplicación OdAJ2EE. OdAJ2EE es una aplicación Java compleja, con alrededor de 100 páginas Web dinámicamente construidas y un mecanismo de persistencia basado en JPA. Por tanto, se decidió implementar la aplicación siguiendo la arquitectura multicapa, pudiendo desarrollar cada capa independientemente y en cualquier orden. Se decidió diseñar cada capa como un paquete UML independiente empezando por la capa de negocio, la capa de integración y finalmente la capa de presentación. Respecto a las capas de negocio e integración, se utilizó UML estándar para especificar su estructura y comportamiento. Se utilizaron diagramas UML de clases para definir los objetos de negocio y los servicios de aplicación, y se utilizaron diagramas UML de secuencia para especificar su comportamiento. De esta forma, se pudieron incluir en los diseños UML patrones arquitectónicos y orientadas a objetos. Respecto a la capa de presentación, los perfiles WAE4x fueron desplegados en la herramienta CASE Borland Together, la cual, junto con Eclipse fue usada a través de todo el proceso de desarrollo de la aplicación. Cada caso de uso se especificó a través de un flujo navegacional modelado con un diagrama WAE4x. Por tanto, los perfiles WAE4x permitieron modelar la capa de presentación Web, que de otra forma podría haberse difuminado en los diseños UML estándar, y aprovechar la expresividad de UML para modelar la integración de los componentes de la capa de presentación con los componentes de las otras capas de la aplicación siguiendo los patrones multicapa.

Se comprobó la expresividad de estos perfiels, ya que con pocos elementos fue posible desarrollar mapas navegacionales de cualquier complejidad. Además, estos elementos

fueron intuitivos de usar, o al menos fáciles de entender para desarrolladores con conocimientos de los frameworks JSF o ASP.NET MVC. Estos componentes permitieron modelar el flujo navegacional sin ambigüedad a través de la integración con los componentes de negocio.

El desarrollo de los modelos WAE4x, permite olvidar las complejidades de los frameworks y el código centrándose en elementos gráficos y sus relaciones. En este sentido, los perfiles son considerados como una nueva capa de abstracción bien definida que permite especificar el flujo navegacional de forma rápida y flexible. El proyecto Java fue generado a través de la ejecución automática de las reglas de transformación, lo cual facilitó la transición de modelos a código. El proyecto junto con el código fue importado directamente en Eclipse permitiendo ser complementado por los desarrolladores con el código de interfaz de usuario y el código que implementó las reglas de negocio.

En OdAJ2EE también se comprobó la escalabilidad de los perfiles WAE4x. Como en el caso de UML, los modelos WAE4x incluyen componentes definidos en diferentes paquetes con enlaces navegacionales entre ellos, de esta forma, los modelos se mantienen en un tamaño razonable. Además, en la generación de código, los diferentes paquetes son traducidos a estructuras de directorios con sus respectivas páginas y clases. OdAJ2EE fue implementada en J2EE, sin embargo no hay diferencias significativas entre el uso de WAE4JSF y WAE4.NET más allá, por supuesto, de las impuestas por los respectivos frameworks. No obstante, al igual que UML-WAE, los modelos desarrollados con los perfiles WAE4x son adecuados sólo para los desarrolladores.

Como el uso de PIMs es valioso en la aproximación MDD, se evaluó la posibilidad de utilizar NMMp como notación de alto nivel para generar a partir de ella los modelos WAE4x. Sin embargo, NMMp define el concepto de anclas que oculta todos los componentes funcionales del lado del servidor. En NMMp estos componentes se generan por las reglas de transformación que convierten los modelos NMMp en modelos UML-WAE. Debido a que uno de los objetivos del presente trabajo es definir transformaciones directas que faciliten la trazabilidad en el proceso MDD, precisando y simplificando lo más posible los componentes fuente y destino en los procesos de transformación, asegurando por otra parte su implementabilidad, fue necesario buscar una nueva alternativa a NMMp.

Esta nueva notación, llamada E-WAE, permite caracterizar mapas navegacionales utilizando conceptos propios de aplicaciones empresariales modernas, preservando los

beneficios de NMMp. E-WAE añade un nivel de abstracción sobre los perfiles WAE4x. De esta forma, de acuerdo al enfoque MDA, E-WAE permite generar modelos PIM que facilitan la reusabilidad, portabilidad y comunicación entre usuarios finales y desarrolladores. Estos pueden transformarse a modelos WAE4x que constituyen modelos PSM que proporcionan una semántica orientada a código a los modelos E-WAE, pueden ser integrados con otros modelos UML a través de patrones arquitectónicos y de diseño, y pueden ser transformados a código de frameworks específicos de forma directa. Además, dado que los modelos WAE4x son PSMs, los desarrolladores pueden usarlos en herramientas UML CASE genéricas para mantener el código que puede ser extendido durante la evolución de las aplicaciones.

En síntesis, el enfoque propuesto por el presente trabajo permite perfilar las funcionalidades soportadas por una aplicación empresarial moderna a través del desarrollo de su capa de presentación Web siguiendo la filosofía de UML. Los modelos E-WAE son modelos PIM bien definidos que permiten la generación de mockups navegacionales que, ejecutándose independientemente de la lógica de negocio, permiten la validación temprana a nivel de usuario de la capa de presentación Web. Esta característica es muy importante debido a que los altos costos del desarrollo del software son originados en su mayoría por la falta de entendimiento de los problemas del dominio que las aplicaciones intentan solucionar.

El uso de mockups de presentación como herramienta para refinar los requisitos de las aplicaciones finales ha sido reconocido tanto en el sector académico como en el industrial. Prueba de ello es el número de propuestas que desarrollan métodos basados en mockups para extender los enfoques orientados al modelado y el alto número de herramientas comerciales disponibles en el mercado para generar esquemas de interfaz de usuario. Sin embargo, estos mockups son rápidamente desarrollados sin tener en cuenta los requisitos no funcionales, y por tanto, con la idea de no ser reutilizados en el proceso de desarrollo de las aplicaciones finales.

El enfoque presentado, como los procesos ágiles, promueve el desarrollo a través de ciclos iterativos cortos involucrando a los usuarios finales en cada iteración. Como los procesos tradicionales, usa la notación UML para definir mapas navegacionales, los cuales actúan como modelos fuente a partir de los cuales se generan los mockups navegacionales siguiendo la filosofía del desarrollo dirigido por modelos (MDD).

Por tanto, el enfoque también puede ser visto como un intento de reconciliar el diseño de la arquitectura y los procesos ágiles. De hecho, no hay mucha investigación y literatura en este respecto (Breivold et al., 2010; Buschmann & Henney, 2013). El diseño de la arquitectura implica invertir esfuerzo y tiempo suficiente para definir qué componentes forman el sistema, cómo se relacionan entre ellos y cómo mediante su interacción cumplen con los requisitos funcionales especificados, cumpliendo además con los requisitos no funcionales como rendimiento, seguridad, robustez, disponibilidad o usabilidad. Sin embargo, este diseño de arquitectura y su posterior mantenimiento causa un conflicto con los procesos ágiles que promueven las iteraciones de desarrollo rápidas para mantener la alta interacción con los usuarios.

Los mockups navegacionales desarrollados con el enfoque aquí presentado pueden ser completamente reutilizados en las aplicaciones finales que implementan arquitecturas concretas. Por medio de las transformaciones automáticas, los mockups pueden ser generados conjuntamente con los usuarios finales, siguiendo de esta forma los principios de las metodologías ágiles. Además, dado que el enfoque propuesto está basado en UML y comprende la definición de modelos PSM, estos pueden ser optimizados o enriquecidos por los desarrolladores a través del uso de patrones arquitectónicos y de diseño. Por tanto, el enfoque confía el desarrollo a los programadores y no está limitado por herramientas específicas que generen el código y acoplen su mantenimiento a generadores propietarios de código. Así se puede afirmar que el enfoque propuesto no sólo se limita a la producción de diseños sino que asiste el proceso de desarrollo sin necesidad de herramientas específicas.

6.2 Trabajo futuro

Aunque los modelos desarrollados con el enfoque presentado se basan en UML, pudiendo ser generados y mantenidos con herramientas UML CASE genéricas, y las reglas de transformación de los modelos PIM E-WAE a los modelos PSM WAE4x se han implementado con el lenguaje estándar QVT *operational mappings*, la generación de código ejecutable requiere del motor de transformaciones M2T específico de Borland Together, así como de una aplicación específicamente desarrollada para desplegar el código final en los IDEs respectivos. De esta forma, aunque Borland Together proporciona un entorno integrado para desarrollar el enfoque presentado, la generación automática de código depende de la herramienta. Por otro lado, estamos convencidos que el desarrollo basado en estándares facilita la portabilidad y el uso de herramientas

genéricas. Por tanto, un objetivo inmediato del presente trabajo es implementar las reglas de transformación explicitas de los modelos WAE4x a código en un lenguaje estándar, como el lenguaje MOFM2T especificado por la OMG (OMG, 2011).

El hecho de que los modelos desarrollados con el enfoque presentado estén basados en UML, le da un alto grado de flexibilidad en sus posibilidades de integración. Por ejemplo, los modelos PIM E-WAE pueden integrarse directamente con modelos PSM que representen otros sistemas o componentes de otras capas de la aplicación. Estos modelos PSM deberían entonces, integrarse con los modelos PSM WAE4x, redefiniendo adecuadamente las relaciones entre los elementos. Otra alternativa podría ser que los modelos PIM E-WAE se integrasen con otros modelos PIM, los cuales deberían entonces, ser transformados a modelos PSM que se integren adecuadamente con los modelos PSM WAE4x. Estudiar estas posibilidades de integración del enfoque presentado, preservando sus características, con componentes de otras capas de las aplicaciones empresariales, e incluso, la traducción de E-WAE a otras notaciones, forma parte del trabajo futuro.

Como los enfoques orientados al modelado, el enfoque presentado esta soportado por una notación bien definida. Por tanto, es factible, aplicando ingeniería inversa, construir los modelos a partir del código generado. El desarrollo de los mecanismos que permitan la aplicación de esta ingeniería inversa también es parte del trabajo futuro.

Finalmente, los perfiles WAE4x soportan el desarrollo de la capa de presentación Web con los frameworks JSF y ASP.NET MVC. Sin embargo, el enfoque ha sido desarrollado de forma que puede ser fácilmente extendido para soportar otros frameworks de capa de presentación y para incluir otras características. Como parte del trabajo futuro está ampliar el catálogo de frameworks soportados, así como enriquecer la definición de los perfiles para soportar características RIA y comunicaciones asíncronas en general.

7 BIBLIOGRAFÍA

Abrahão, S., Olsina, L., & Pastor, O. (2002, October). Towards the quality evaluation of functional aspects of operative web applications. In *International Conference on Conceptual Modeling* (pp. 325-338). Springer Berlin Heidelberg.

Acerbis, R., Bongio, A., Brambilla, M., Butti, S., Ceri, S., & Fraternali, P. (2008, June). Web applications design and development with webml and webratio 5.0. In *International Conference on Objects, Components, Models and Patterns* (pp. 392-411). Springer Berlin Heidelberg

Alur, D., Crupi, J., & Malks, D. (2003). *Core J2EE Patterns. Best Practices and Design Strategies*: Sun Microsystems Press/Prentice Hall.

Arlow, J., and Neustadt, I. (2005) UML 2 and the Unified Process. Practical Object-Oriented Analysis and Design. 2nd Edition: Addisson-Wesley Professional

Barry, C., and Lang. M. (2001) A survey of multimedia and web development techniques and methodology usage.

Berner, S., Glinz, M., & Joos, S. (1999, October). A classification of stereotypes for object-oriented modeling languages. In *International Conference on the Unified Modeling Language* (pp. 249-264). Springer Berlin Heidelberg.

Booch, G., Maksimchuk, R., Engel, M., Young, B., Conallen, J. & Houston, K. 2007. *Object-Oriented Analysis and Design with Applications,* Addison-Wesley.

Bourque, P., Fairley, P. (2014). SWEBOK v3. Guide to the Software Egineering Body of Knowledge. *IEEE Computer Society*.

Bozzon, A., Comai, S., Fraternali, P., & Carughi, G. T. (2006, July). Conceptual modeling and code generation for rich internet applications. In *Proceedings of the 6th international conference on Web engineering* (pp. 353-360). ACM.

Brambilla, M., Butti, Stefano. (2012). *From WebML to WebRatio and IFML*. http://dbgroup.como.polimi.it/brambilla/sites/dbgroup.como.polimi.it.brambilla/files/referencematerials/WebML-IFML-WebRatio-Novatica-EN.pdf

Brambilla, M., Ceri, S., Facca, F. M., Celino, I., Cerizza, D., & Valle, E. D. (2007). Model-driven design and development of semantic Web service applications. *ACM Transactions on Internet Technology* (TOIT), 8(1), 3.

Brambilla, M., Ceri, S., Fraternali, P., & Manolescu, I. (2006). Process modeling in web applications. *ACM Transactions on Software Engineering and Methodology* (TOSEM), 15(4), 360-409.

Breivold, H. P., Sundmark, D., Wallin, P., & Larsson, S. (2010, August). What does research say about agile and architecture?. In *Software Engineering Advances* (ICSEA), *2010 Fifth International Conference on* (pp. 32-37). IEEE.

Brown, S., Burdick, R., Falkner, J., Galbraith, B., Johnson, R., Kim, L., Kochmer, C., Kristmudsson, T. & Li, S. 2002. *Professional JSP*, Birmingham.

Brown, S., Dalton, S., Jepp, D., Johnson, D., Li, S. & Raible, M. 2005. *Pro JSP 2*, Berkeley, CA.

Brucker, A. D., & Doser, J. (2007). Metamodel-based UML notations for domain-specific languages. In *4th International Workshop on Software Language Engineering (ATEM 2007).*

Busch, M., Knapp, A., & Koch, N. (2011). Modeling Secure Navigation in Web Information Systems Perspectives in Business Informatics Research (pp. 239-253): Springer.

Busch, M., Koch, N., Masi, M., Pugliese, R., & Tiezzi, F. (2012). *Towards model-driven development of access control policies for web applications.* Paper presented at the Proceedings of the Workshop on Model-Driven Security.

Buschmann, F., & Henney, K. (2013). Architecture and agility: Married, divorced, or just good friends?. *IEEE software*, 30(2), 80-82.

Cachero, C., (2003). OO-H: Una extensión a los métodos OO para el modelado y generación automática de interfaces hipermediales (Doctoral dissertation, Universidad de Alicante).

Ceri, S., Fraternali, P., & Matera, M. (2002). Conceptual modeling of data-intensive Web applications. *IEEE Internet Computing*, 6(4), 20-30.

Chadwick, J. (2011). *Programming Razor*. Sebastopol, CA :: O'Reilly.

Comai, S., & Fraternali, P. (2001, July). A semantic model for specifying data-intensive Web applications using WebML. In SWWS (pp. 566-585).

Conallen, J. (1999). Modeling Web application architectures with UML. *Communications of the ACM*, 42(10), 63-70.

Cortés, H. (2011). Extensión UML para el modelado de mapas navegacionales de aplicaciones web basado en MDA. [Trabajo fin de Máster]

Cortés, H., & Navarro, A. (2014). NMMp: A Model-Driven UML Extension for the Description of Navigation Maps for Web Applications. *International Journal of Software Engineering and Knowledge Engineering*, 24(03), 391-417. doi:10.1142/S0218194014500156

Cortés, H., & Navarro, A. (2016). MDD Inclusion of Navigational, Structural and RBAC Elements for JSF and ASP.NET MVC Frameworks in UML Models. *Fourth International Conference on Technological Ecosystems for Enhancing Multiculturality*, Salamanca, Spain, November 02 - 04, 2016. ACM Press.

Crawford, W., & Kaplan, J. (2003). *J2EE design patterns.* " O'Reilly Media, Inc.".

Erl, T. (2007). SOA: *Principles of Service Design*: Prentice-Hall.

Esposito, D. (2011). *Programming Microsoft ASP.NET MVC* (Vol. 2nd ed). Microsoft Press. Redmond, Washington

Esposito, D. (2014). Microsoft.NET: *architecting applications for the enterprise* (Second ed.). Redmond: Microsoft Press.

Esposito, D. (2016). Modern Web Development: Understanding domains, technologies, and user experience. Microsoft Press.

Etzion, O. & Niblett, P. *Event Processing in Action*. Manning Publications

Fuentes, L., & Vallecillo, A. (2004). *Una introducción a los perfiles UML*. University of Malaga, March.

Fowler, M. (2002). Patterns of enterprise application architecture. Boston: Addison-Wesley.

Gamma, E., Vlissides, J., Helm, R., Johnson, R. (1995). Design patterns: Elements of reusable object-oriented software. *Reading: Addison-Wesley*, *49*(120), 11.

Gardner, T., & Yusuf, L. (2006). Explore model-driven development (MDD) and related approaches: A closer look at model-driven development and other industry initiatives. IBM *developerworks, March*.

Geary, D.M., & Horstmann, C.S. (2010). *Core JavaServer Faces. 3rd edition*: Prentice-Hall.

Gómez, J., Cachero, C., & Pastor, O. (2001). On Conceptual Modeling of Device-Independent Web Applications: Towards a Web-Engineering Approach. *Ieee multimedia*, 8(2), 26-39.

Han, M., & Hofmeister, C. (2005, July). Separation of navigation routing code in J2EE web applications. In *International Conference on Web Engineering* (pp. 221-231). Springer Berlin Heidelberg.

Hansen, M.D. (2007). *SOA* Using Java Web Services: Prentice Hall.

Isakowitz, T., Stohr, E. A., & Balasubramanian, P. (1995). RMM: a methodology for structured hypermedia design. *Communications of the ACM*, 38(8), 34-44.

Jacobson, I., Rumbaugh, J., and G. Booch. (1999). *The Unified Software Development Process*. Addison-Wesley, Reading, Massachusetts.

Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., & Valduriez, P. (2006, October). ATL: a QVT-like transformation language. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications* (pp. 719-720). ACM.

Jung, H. W., Kim, S. G., & Chung, C. S. (2004). *Measuring software product quality*: A survey of ISO/IEC 9126. *IEEE software*, 21(5), 88-92. Doi: http://dx.doi.org/10.1109/MS.2004.1331309

Keith, M., & Schincariol, M. (2013). Pro *JPA 2*: Apress.

Koch, N., & Kraus, A. (2002, June). The expressive power of uml-based web engineering. In *Second International Workshop on Web-oriented Software Technology* (IWWOST02) (Vol. 16). CYTED.

Kroiss, C., Koch, N., & Knapp, A. (2009, June). Uwe4jsf: A model-driven generation approach for web applications. In *International Conference on Web Engineering* (pp. 493-496). Springer Berlin Heidelberg.

Lerman, J. (2012). *Programming entity framework* Code First (Vol. 1st ed). Sebastopol, CA :: O'Reilly Media.

Lowy, J. (2010). Programming *WCF* services (Vol. 3rd ed). Farnham :: O'Reilly.

Manolescu, I., Brambilla, M., Ceri, S., Comai, S., & Fraternali, P. (2005). Model-driven design and deployment of service-enabled web applications. *ACM Transactions on Internet Technology* (TOIT), 5(3), 439-479.

Meier, J. D. (2006). Web application security engineering. Security & Privacy, *IEEE*, 4(4), 16-24. Doi: 10.1109/MSP.2006.109

Meliá, S., Gómez, J., Pérez, S., & Díaz, O. (2008, July). A model-driven development for GWT-based Rich Internet Applications with OOH4RIA. In *Web Engineering, 2008. ICWE'08. Eighth International Conference on* (pp. 13-23). IEEE.

Meliá, S., Domene, J. J. M., Pérez, Á., & Gómez, J. (2009). OOH4RIA Tool: Una Herramienta basada en el Desarrollo Dirigido por Modelos para las RIAs. In *JISBD* (pp. 219-222).

Mendes, R. (2016). *The 13 Commandments of DevOps*. https://www.outsystems.com/blog/2016/02/the-13-commandments-of-devops.html

Mendix (2016b). *Mendix rapid application development model stages*. Mendix. https://www.mendix.com/rapid-application-development-model/

Microsoft. (2014). Microsoft by the Numbers. A collection of statistics bout Microsoft products and services. https://news.microsoft.com/bythenumbers/ms_numbers.pdf

Moreno, N., Fraternali, P., & Vallecillo, A. (2007). WebML modelling in UML. *IET software*, 1(3), 67-80.

Navarro, A., & Cortés, H. (2011). Metamodeling or profiling: a practical case in the web engineering domain. In XVII *Congreso Argentino de Ciencias de la Computación.*

Navarro, A., Cristobal, J., Fernández-Chamizo, C., Fernández-Valmayor, A. (2012) *Architecture of a multiplatform virtual campus. Software: Practice and Experience* 42(10), 1229-1246.

Navarro, A., Merino, J., Fernández-Valmayor, A., & Cristobal, J. (2008b). Translating Workflow Diagrams into Web Designs. In *SEKE* (pp. 667-672).

Navarro, A., Cesteros, A. M. F. P., Fernández-Chamizo, C., & Fernández-Valmayor, A. (2013). A Meta-Relational Approach for the Definition and Management of Hybrid Learning Objects. *Educational Technology & Society,* 16(4), 258-274.

Navarro, A., Fernández-Valmayor, A., Fernández-Manjón, B., & Sierra, J. L. (2008). Characterizing navigation maps for web applications with the NMM approach. *Science of Computer Programming*, 71(1), 1-16, 2008. doi:10.1016/j.scico.2007.10.003.

Natis, Y.V., Pezzini, M., Iijima, K., Thomas, A. Dunie, R. (2015) *Magic Quadrant for Enterprise Application Platform as a service*, Worldwide. Garner, Inc.

OMG, (2003). *MDA Guide Version*. http://www.omg.org

OMG, (2008). Meta object facility (mof) 2.0 query/view/transformation specification. *Final Adopted Specification (November 2005).* http://www.omg.org

OMG, (2010). *UML Infrastructure Specification*. http://www.omg.org

OMG, (2011). MOF Model To Text Transformation Language. http://www.omg.org

OMG, (2013). *Semantics of a Foundational Subset for Executable UML Models* (fUML). http://www.omg.org

OMG, (2015). *Interaction Flow Modeling Language* (IFML). http://www.omg.org/spec/IFML/

OpenLanzlo, (2010) . http://www.openlaszlo.org/

Oracle, (2007). Java Authentication and Authorization Service (JAAS) Reference Guide https://docs.oracle.com/javase/8/docs/technotes/guides/security/jaas/JAASRefGuide.html

Pastor, O., Insfrán, E., Pelechano, V., Romero, J., & Merseguer, J. (1997, June). OO-Method: an OO software production environment combining conventional and formal methods. In International Conference on Advanced Information Systems Engineering (pp. 145-158). Springer Berlin Heidelberg.

Pérez-Martínez, J. E. (2003). Heavyweight extensions to the UML metamodel to describe the C3 architectural style. *ACM SIGSOFT Software Engineering Notes*, 28(3), 5-5.

Pronschinske. (2015). *JSF and Spring MVC* tien in Java/JVM Frameworks Poll. Java Zone. https://dzone.com/articles/poll-what-java-jvm-frameworks-do-you-use

Rumbaugh, J., Jacobson, I. & Booch, G. 2010. *Unified Modeling Language Reference Manual*, Addison-Wesley.

Reilly, D. J. (2006). *Programming Microsoft web forms:* Y1 - 2006

Ricca, F., Scanniello, G., Torchiano, M., Reggio, G., & Astesiano, E. (2010, September). On the effectiveness of screen mockups in requirements engineering: results from an internal replication. In *Proceedings of the 2010 ACM-IEEE international symposium on empirical software engineering and measurement* (p. 17). ACM.

Rivero, J. M., Rossi, G., Grigera, J., Burella, J., Luna, E. R., & Gordillo, S. (2010, July). From mockups to user interface models: an extensible model driven approach. In *International Conference on Web Engineering* (pp. 13-24). Springer Berlin Heidelberg.

Rivero, L., Marczak, S., & Conte, T. (2013). An Approach for the Elicitation of Usability Requirements in the Development of Web Applications. In *CEUR Workshop Proceeding of Requirements Engineering*.

Rivero, J. M., Grigera, J., Rossi, G., Luna, E. R., Montero, F., & Gaedke, M. (2014). Mockup-driven development: providing agile support for model-driven web engineering. *Information and Software Technology*, 56(6), 670-687.

Rivero, J. M., Rossi, G., Grigera, J., Luna, E. R., & Navarro, A. (2011). From interface mockups to web application models. *Paper presented at the International Conference on Web Information Systems Engineering*.

Rossi, G., Pastor, Ó., Schwabe, D., & Olsina, L. (Eds.). (2007). *Web engineering: modelling and implementing web applications*. Springer Science & Business Media.

Rymer, J.R., Richardson, C., Mines, C., Jones, D., Whittaker, D., Miller, J. & McPherson, I. Low-Code Platforms Deliver Customer-Facing Apps Fast, But Will They Scale Up?. *Forrester Research*

Steel, C., Nagappan, R. and Lai, R.. Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management. Prentice Hall, 2005.

Sommerville, I. (2010). *Software Engineering*. Addison Wesley.

Stober, T., & Hansmann, U. (2010). Agile Software Development: Best Practices for Large Software Development Projects

Stotts, P. D., Furuta, R., and Ruiz, C., (1998). Hyperdocuments as automata: Verification of trace-based browsing properties by model checking, *ACM Transactions on Information Systems* 16 1–30.

Thakurta, R., & Ahlemann, F. (2011). *Understanding Requirement Volatility In Software Projects*. Engineering Management Journal, 23(3), 3-7.

Thomas, Y.V Ntias, M. Pezzini, and K. Iijima. Market Guide: *On-Premises Application Platforms. Gartner*, Inc, 2015. https://www.gartner.com/doc/3172019?srcId=1-2819006590

WinterGreen. (2014) WinterGreen Research Inc. Report #SH26021314. Application Server Market Shares, Strategies, and Forecasts, Worldwide, 2014 to 2020. http://wintergreenresearch.com/reports/application%20server.html

Weisemöller, I., & Schürr, A. (2007, September). A comparison of standard compliant ways to define domain specific languages. International Conference on Model Driven Engineering Languages and Systems (pp. 47-58). Springer Berlin Heidelberg.

8 ANEXOS

8.1 WAE4JSF a código

8.1.1 Generación páginas JSF

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns:xta="http://www.borland.com/xta"
          xsi:schemaLocation="http://www.w3.org/1999/XSL/Transform
       platform:/plugin/com.borland.tg.xsl.core/schema/schema-for-xslt20.xsd">
    <xta:metamodel uri="http://www.borland.com/together/uml"/>
    <xta:metamodel uri="http://www.borland.com/together/uml20"/>


        <xsl:template name="WebPagesGeneration"
        match="self.oclIsKindOf(uml::together::Model)">
                <!-- For each package apply this rule -->
                <xsl:apply-templates select="self.ownedMembers->select(o|
                o.oclIsKindOf(uml::kernel::packages::Package))"
                mode="WebPagesGeneration.Packages"/>
    </xsl:template>

        <xsl:template mode="WebPagesGeneration.Packages"
        match="self.oclIsKindOf(uml::kernel::packages::Package)">
                <!-- For each package apply this rule -->
                <xsl:element name="Folder">
                        <xsl:attribute name="Name">
                                <xsl:value-of select="self.name"/>
                        </xsl:attribute>

                <!-- For each "JSF Server Page" class apply this rule -->
                <xsl:apply-templates select="self.ownedMembers->select(o
                |o.oclIsKindOf(uml20::classes::Class) and o.stereotypes = OrderedSet{'JSF
                Server Page'})" mode="WebPagesGeneration.JSF"/>

                <!-- For each "Template" class apply this rule -->
                <xsl:apply-templates select="self.ownedMembers->select(o
                |o.oclIsKindOf(uml20::classes::Class) and o.stereotypes =
                OrderedSet{'Template'})" mode="WebPagesGeneration.Template"/>

                <!-- For each "Template Component" class apply this rule -->
                <xsl:apply-templates select="self.ownedMembers->select(o
                |o.oclIsKindOf(uml20::classes::Class) and o.stereotypes =
                OrderedSet{'Template Component'})"
                mode="WebPagesGeneration.TemplateComponent"/>

                <xsl:apply-templates select="self.ownedMembers->select(o |
                o.oclIsKindOf(uml::kernel::packages::Package))"
                mode="WebPagesGeneration.Packages"/>

          </xsl:element>
        </xsl:template>

        <xsl:template mode="WebPagesGeneration.JSF"
        match="self.oclIsKindOf(uml20::classes::Class) and self.stereotypes =
        OrderedSet{'JSF Server Page'}">
                <!-- For each "JSF Server Page" class apply this rule -->
                <xsl:element name="File">
                        <xsl:attribute name="Name">
                                <xsl:value-of select="self.name +'.xhtml'"/>
                        </xsl:attribute>
                        <xsl:attribute name="Type">
                                <xsl:value-of select="'Content'"/>
                        </xsl:attribute>

                        <xsl:element name="Content">

                                <xsl:element name="html">
                                        <xsl:element name="f:view">
                                                <xsl:element name="body">

                                                        <!-- If the current page uses a
                                                        template page -->
                                                        <xsl:choose>
```

```
                                                        <xsl:when
                                                        test="self.dependencies-
                                                        >select(o | o.stereotypes =
                                                        OrderedSet{'Use'})-
                                                        >exists(true)">
                                                                <xsl:element
                                                                name="ui:composition"
                                                                >
                                                                <xsl:attribute
                                                                name="template">
                                                                <xsl:value-of
                                                                select="replace(self.
                                                                dependencies-
                                                                >select(o |
                                                                o.stereotypes =
                                                                OrderedSet{'Use'})-
                                                                >any(true)->collect(d
                                                                | d.supplier)-
                                                                >any(true).oclAsType(
                                                                uml20::classes::Class
                                                                ).fullName, '.',
                                                                '/')"/>

                                                        </xsl:attribute>
                                                        <xsl:element
                                                        name="ui:define">
                                                        <xsl:attribute name="name">
                                                                <xsl:value-of
                                                                select="self.getPrope
                                                                rtyValue('ProfileWAE_
                                                                4_JSF_x__y_JSF_Server
                                                                _Page_x__y_DefaultTem
                                                                plateComponent')"/>

                                                </xsl:attribute>

                                                <!-- To create the JSF Content apply
                                                this rule -->
                                                <xsl:apply-templates select="self"
                                                mode="WebPagesGeneration.JSFContent"
                                                />


                                                        </xsl:element>
                                                </xsl:element>
                                        </xsl:when>
                                        <xsl:otherwise>
                                                <!-- To create the JSF Content apply this rule -->
                                                <xsl:apply-templates select="self"
                                                mode="WebPagesGeneration.JSFContent"/>
                                        </xsl:otherwise>
                                </xsl:choose>
                        </xsl:element>
</xsl:element>
</xsl:element>
</xsl:element>

</xsl:element>
</xsl:template>

<xsl:template mode="WebPagesGeneration.JSFContent"
match="self.oclIsKindOf(uml20::classes::Class) and self.stereotypes = OrderedSet{'JSF
Server Page'}">
                <!-- For each "Div" class apply this rule -->
                <xsl:apply-templates select="self.associations->collect(d | d.supplier)"
                mode="WebPagesGeneration.JSFDiv"/>

                <!-- For each "Form" class apply this rule -->
                <xsl:apply-templates select="self.associations->collect(d | d.supplier)"
                mode="WebPagesGeneration.JSFForms"/>

                <!-- For each "Static Link" dependency apply this rule -->
                <xsl:apply-templates select="self.dependencies->select(o | o.stereotypes
                = OrderedSet{'Static Link'})" mode="WebPagesGeneration.StaticLink"/>

                <!-- For each "Dynamic Link" dependency apply this rule -->
```

```
        <xsl:apply-templates select="self.dependencies->select(o | o.stereotypes
        = OrderedSet{'Dynamic Link'})" mode="WebPagesGeneration.DynamicLink"/>
</xsl:template>

<xsl:template mode="WebPagesGeneration.JSFDiv"
match="self.oclIsKindOf(uml20::classes::Class) and self.stereotypes =
OrderedSet{'JSF Div'}">
        <!-- For each "JSF Div" class apply this rule -->
        <xsl:element name="div">
                <xsl:attribute name="id">
                        <xsl:value-of select="self.name"/>
                </xsl:attribute>
                <!-- For each nested "Div" class apply this rule -->
                <xsl:apply-templates select="self.associations->collect(d |
                d.supplier)" mode="WebPagesGeneration.JSFDiv"/>
                <!-- For each "Form" class apply this rule -->
                <xsl:apply-templates select="self.associations->collect(d |
                d.supplier)" mode="WebPagesGeneration.JSFForms"/>
                <!-- For each "Static Link" dependency apply this rule -->
                <xsl:apply-templates select="self.dependencies->select(o |
                o.stereotypes = OrderedSet{'Static Link'})"
                mode="WebPagesGeneration.StaticLink"/>
                <!-- For each "Dynamic Link" dependency apply this rule -->
                <xsl:apply-templates select="self.dependencies->select(o |
                o.stereotypes = OrderedSet{'Dynamic Link'})"
                mode="WebPagesGeneration.DynamicLink"/>
        </xsl:element>
</xsl:template>

<xsl:template mode="WebPagesGeneration.JSFForms"
match="self.oclIsKindOf(uml20::classes::Class) and self.stereotypes =
OrderedSet{'JSF Form'}">
        <!-- For each "JSF Form" class apply this rule -->
        <xsl:element name="h:form">
                <xsl:attribute name="Name">
                        <xsl:value-of select="self.name"/>
                </xsl:attribute>
                <!-- For each "Outcome" class apply this rule -->
                <xsl:apply-templates select="self.dependencies->select(o |
                o.stereotypes = OrderedSet{'Static Submit'})"
                mode="WebPagesGeneration.StaticSubmit"/>
                <!-- For each "Managed Bean" class apply this rule -->
                <xsl:apply-templates select="self.dependencies->select(o |
                o.stereotypes = OrderedSet{'Dynamic Submit'})"
                mode="WebPagesGeneration.DynamicSubmit"/>
        </xsl:element>
</xsl:template>

<xsl:template mode="WebPagesGeneration.StaticSubmit"
match="self.oclIsKindOf(uml20::classes::Dependency) and self.stereotypes =
OrderedSet{'Static Submit'}">
        <!-- For each "Outcome" class apply this rule -->
        <xsl:element name="h:commandButton">
                <xsl:attribute name="value">
                        <xsl:value-of
                        select="'To'.concat(self.supplier.oclAsType(uml20::classes
                        ::Class).dependencies->select(o | o.stereotypes =
                        OrderedSet{'Access'})->collect(d | d.supplier)-
                        >any(true).oclAsType(uml20::classes::Class).name)"/>
                </xsl:attribute>
                <xsl:attribute name="action">
                        <xsl:value-of
                        select="self.supplier.oclAsType(uml20::classes::Class).nam
                        e"/>
                </xsl:attribute>
        </xsl:element>
</xsl:template>

<xsl:template mode="WebPagesGeneration.DynamicSubmit"
match="self.oclIsKindOf(uml20::classes::Dependency) and self.stereotypes =
OrderedSet{'Dynamic Submit'}">
        <!-- For each "Managed Bean" class apply this rule -->
        <xsl:element name="h:commandButton">
                <xsl:attribute name="value">
                        <xsl:value-of
                        select="'SubmitTo'.concat(self.supplier.oclAsType(uml20::c
                        lasses::Class).name)"/>
                </xsl:attribute>
                <xsl:attribute name="action">
```

```
                        <xsl:value-of
                        select="'&#123;'.concat(self.supplier.oclAsType(uml20::cl
                        asses::Class).name).concat('.').concat(self.getPropertyVal
                        ue('ProfileWAE_4_JSF_x__y_Dynamic_Submit_x__y_Action'))"/>
                </xsl:attribute>
        </xsl:element>
</xsl:template>

<!-- For each "Static Link" dependency apply this rule -->
<xsl:template mode="WebPagesGeneration.StaticLink"
match="self.oclIsKindOf(uml20::classes::Dependency) and self.stereotypes =
OrderedSet{'Static Link'}">
        <!-- For each "Outcome" class apply this rule -->
        <xsl:element name="h:link">
                <xsl:attribute name="value">
                        <xsl:value-of
                        select="'To'.concat(self.supplier.oclAsType(uml20::classes
                        ::Class).dependencies->select(o | o.stereotypes =
                        OrderedSet{'Access'})->collect(d | d.supplier)-
                        >any(true).oclAsType(uml20::classes::Class).name)"/>
                </xsl:attribute>
                <xsl:attribute name="outcome">
                        <xsl:value-of
                        select="self.supplier.oclAsType(uml20::classes::Class).nam
                        e"/>
                </xsl:attribute>
        </xsl:element>
</xsl:template>

<!-- For each "Dynamic Link" dependency apply this rule -->
<xsl:template mode="WebPagesGeneration.DynamicLink"
match="self.oclIsKindOf(uml20::classes::Dependency) and self.stereotypes =
OrderedSet{'Dynamic Link'}">
        <!-- For each "Managed Bean" class apply this rule -->
        <xsl:element name="h:link">
                <xsl:attribute name="value">
                        <xsl:value-of
                        select="'LinkTo'.concat(self.supplier.oclAsType(uml20::cla
                        sses::Class).name)"/>
                </xsl:attribute>
                <xsl:attribute name="outcome">
                        <xsl:value-of
                        select="'&#123;'.concat(self.supplier.oclAsType(uml20::cl
                        asses::Class).name).concat('.get').concat(self.getProperty
                        Value('ProfileWAE_4_JSF_x__y_Dynamic_Link_x__y_Action'))"/
                        >
                </xsl:attribute>
        </xsl:element>
</xsl:template>

<!-- For each "Template" dependency apply this rule -->
<xsl:template mode="WebPagesGeneration.Template"
match="self.oclIsKindOf(uml20::classes::Class) and self.stereotypes =
OrderedSet{'Template'}">
        <!-- For each "JSF Server Page" class apply this rule -->
        <xsl:element name="File">
                <xsl:attribute name="Name">
                        <xsl:value-of select="self.name +'.jsf'"/>
                </xsl:attribute>
                <xsl:attribute name="Type">
                        <xsl:value-of select="'Content'"/>
                </xsl:attribute>

                <xsl:element name="Content">
                        <xsl:element name="html">
                                <xsl:element name="body">
                                        <!-- To create the template Content apply
                                        this rule -->
                                    <xsl:apply-templates
                                    select="self.dependencies->select(d |
                                    d.stereotypes = OrderedSet{'Include'})-
                                    >collect(d | d.supplier)"
                                    mode="WebPagesGeneration.IncludedTemplateCompo
                                    nent"/>
                                </xsl:element>
                        </xsl:element>
                </xsl:element>
```

```
                </xsl:element>
        </xsl:template>

        <!-- For each "Template Component" class to include within the "Template" class
        apply this rule -->
        <xsl:template mode="WebPagesGeneration.IncludedTemplateComponent"
        match="self.oclIsKindOf(uml20::classes::Class) and self.stereotypes =
        OrderedSet{'Template Component'}">
                <xsl:element name="ui:insert">
                        <xsl:attribute name="name">
                                <xsl:value-of select="self.name"/>
                        </xsl:attribute>
                <xsl:element name="ui:include">
                        <xsl:attribute name="src">
                                <xsl:value-of
                                select="'/'.concat(replace(self.fullName,'.','/')).concat(
                                '.jsf')"/>
                        </xsl:attribute>
                </xsl:element>
            </xsl:element>
        </xsl:template>

        <!-- For each "Template Component" class apply this rule -->
        <xsl:template mode="WebPagesGeneration.TemplateComponent"
        match="self.oclIsKindOf(uml20::classes::Class) and self.stereotypes =
        OrderedSet{'Template Component'}">
                <!-- For each "Template Component" class apply this rule -->
                <xsl:element name="File">
                        <xsl:attribute name="Name">
                                <xsl:value-of select="self.name +'.jsf'"/>
                        </xsl:attribute>
                        <xsl:attribute name="Type">
                                <xsl:value-of select="'Content'"/>
                        </xsl:attribute>

                        <xsl:element name="Content">
                                <xsl:element name="html">
                                        <xsl:element name="body">
                                                <xsl:element name="ui:composition">
                                                <!-- For each "Static Link" dependency
                                                apply this rule -->
                                                <xsl:apply-templates
                                                select="self.dependencies->select(o |
                                                o.stereotypes = OrderedSet{'Static Link'})"
                                                mode="WebPagesGeneration.StaticLink"/>


                                                <!-- For each "Dynamic Link" dependency
                                                apply this rule -->
                                                <xsl:apply-templates
                                                select="self.dependencies->select(o |
                                                o.stereotypes = OrderedSet{'Dynamic
                                                Link'})"
                                                mode="WebPagesGeneration.DynamicLink"/>
                                        </xsl:element>
                                        </xsl:element>
                                </xsl:element>
                        </xsl:element>

                </xsl:element>
        </xsl:template>

</xsl:stylesheet>
```

8.1.2 Generación managed beans JSF

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns:xta="http://www.borland.com/xta"
          xsi:schemaLocation="http://www.w3.org/1999/XSL/Transform
        platform:/plugin/com.borland.tg.xsl.core/schema/schema-for-xslt20.xsd">
    <xta:metamodel uri="http://www.borland.com/together/uml"/>
    <xta:metamodel uri="http://www.borland.com/together/uml20"/>
```

```
        <xsl:template name="ManagedBeansGeneration"
        match="self.oclIsKindOf(uml::together::Model)">
                <!-- Generate the structure Src/Presentation/ManagedBeans -->
        <xsl:element name="Folder">
                <xsl:attribute name="Name">
                        <xsl:value-of select="'Src'"/>
                </xsl:attribute>

                <xsl:element name="Folder">
                        <xsl:attribute name="Name">
                                <xsl:value-of select="'Presentation'"/>
                        </xsl:attribute>

                        <xsl:element name="Folder">
                                <xsl:attribute name="Name">
                                        <xsl:value-of select="'ManagedBeans'"/>
                                </xsl:attribute>

                                <!-- For each package apply this rule -->
                                <xsl:apply-templates select="self.ownedMembers->select(o|
                                o.oclIsKindOf(uml::kernel::packages::Package))"
                                mode="ManagedBeansGeneration.Packages"/>

                        </xsl:element>

                </xsl:element>
            </xsl:element>
    </xsl:template>

    <!-- Generate the folder structure -->
        <xsl:template mode="ManagedBeansGeneration.Packages"
        match="self.oclIsKindOf(uml::kernel::packages::Package)">
        <!-- For each package apply this rule -->
        <xsl:element name="Folder">
                <xsl:attribute name="Name">
                        <xsl:value-of select="self.name"/>
                </xsl:attribute>

                        <!-- Generate Managed-Bean class -->
                        <xsl:apply-templates select="self.ownedMembers->select(o
                        |o.oclIsKindOf(uml20::classes::Class) and o.stereotypes =
                        OrderedSet{'Managed Bean'})"
                        mode="ManagedBeansGeneration.ManagedBean"/>

                        <xsl:apply-templates select="self.ownedMembers->select(o |
                        o.oclIsKindOf(uml::kernel::packages::Package))"
                        mode="ManagedBeansGeneration.Packages"/>

                </xsl:element>
        </xsl:template>

        <!-- Generate the java managed bean files -->
        <xsl:template mode="ManagedBeansGeneration.ManagedBean"
        match="self.oclIsKindOf(uml20::classes::Class) and self.stereotypes =
        OrderedSet{'Managed Bean'}">
                <xsl:element name="File">
                        <xsl:attribute name="Name">
                                <xsl:value-of select="self.name.concat('.java')"/>
                        </xsl:attribute>
                        <xsl:attribute name="Type">
                                <xsl:value-of select="'ClassType'"/>
                        </xsl:attribute>

                        <xsl:element name="Content">
                                <xsl:element name="Class">
                                        <xsl:attribute name="Name">
                                                <xsl:value-of select="self.name"/>
                                        </xsl:attribute>
                                </xsl:element>
                        </xsl:element>
                </xsl:element>
        </xsl:template>

</xsl:stylesheet>
```

8.1.3 Generación métodos managed bean JSF

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns:xta="http://www.borland.com/xta"
          xsi:schemaLocation="http://www.w3.org/1999/XSL/Transform
        platform:/plugin/com.borland.tg.xsl.core/schema/schema-for-xslt20.xsd">
    <xta:metamodel uri="http://www.borland.com/together/uml"/>
    <xta:metamodel uri="http://www.borland.com/together/uml20"/>


        <xsl:template name="ManagedBeanMethodsGeneration"
        match="self.oclIsKindOf(uml::together::Model)">
                <xsl:element name="ManagedBeans">
                        <!-- For each package apply this rule -->
                        <xsl:apply-templates select="self.ownedMembers->select(o|
                        o.oclIsKindOf(uml::kernel::packages::Package))"
                        mode="ManagedBeanMethodsGeneration.Packages"/>
            </xsl:element>
    </xsl:template>

    <!-- Search the methods for each managed bean -->
        <xsl:template mode="ManagedBeanMethodsGeneration.Packages"
        match="self.oclIsKindOf(uml::kernel::packages::Package)">
                <!-- For each "Form"-"Dynamic Submit" relationship apply this rule -->
                <xsl:apply-templates select="self.ownedMembers->select(o
                |o.oclIsKindOf(uml20::classes::Class) and o.stereotypes = OrderedSet{'JSF
                Form'})" mode="ManagedBeanMethodsGeneration.JSFForm"/>

                <!-- For each "JSF"-"Dynamic Link" relationship apply this rule -->
                <xsl:apply-templates select="self.ownedMembers->select(o
                |o.oclIsKindOf(uml20::classes::Class) and o.stereotypes = OrderedSet{'JSF
                Server Page'})" mode="ManagedBeanMethodsGeneration.JSF"/>

                <!-- For each "TemplatedComponent"-"Dynamic Link" relationship apply this
                rule -->
                <xsl:apply-templates select="self.ownedMembers->select(o
                |o.oclIsKindOf(uml20::classes::Class) and o.stereotypes =
                OrderedSet{'Template Component'})"
                mode="ManagedBeanMethodsGeneration.TemplateComponent"/>

                <xsl:apply-templates select="self.ownedMembers->select(o |
                o.oclIsKindOf(uml::kernel::packages::Package))"
                mode="ManagedBeanMethodsGeneration.Packages"/>
        </xsl:template>

        <xsl:template mode="ManagedBeanMethodsGeneration.JSFForm"
        match="self.oclIsKindOf(uml20::classes::Class) and self.stereotypes =
        OrderedSet{'JSF Form'}">

        <xsl:apply-templates select="self.dependencies->select(o | o.stereotypes =
        OrderedSet{'Dynamic Submit'})"
        mode="ManagedBeanMethodsGeneration.DynamicSubmit"/>

        </xsl:template>

        <!-- Generate the java managed bean files -->
        <xsl:template mode="ManagedBeanMethodsGeneration.DynamicSubmit"
        match="self.oclIsKindOf(uml20::classes::Dependency)">
                <xsl:element name="ManagedBean">
                        <xsl:attribute name="Name">
                                <xsl:value-of
                                select="self.supplier.oclAsType(uml20::classes::Class).nam
                                e"/>
                        </xsl:attribute>
                        <xsl:element name="Method">
                                <xsl:attribute name="Name">
                                        <xsl:value-of
                                        select="self.getPropertyValue('ProfileWAE_4_JSF_x__
                                        y_Dynamic_Submit_x__y_Action')"/>
                                </xsl:attribute>
                    </xsl:element>
                </xsl:element>
        </xsl:template>
```

```
        <xsl:template mode="ManagedBeanMethodsGeneration.JSF"
        match="self.oclIsKindOf(uml20::classes::Class) and self.stereotypes =
        OrderedSet{'JSF Server Page'}">

                <xsl:apply-templates select="self.dependencies"
                mode="ManagedBeanMethodsGeneration.DynamicLink"/>

        </xsl:template>

        <xsl:template mode="ManagedBeanMethodsGeneration.TemplateComponent"
        match="self.oclIsKindOf(uml20::classes::Class) and self.stereotypes =
        OrderedSet{'Template Component'}">

                <xsl:apply-templates select="self.dependencies"
                mode="ManagedBeanMethodsGeneration.DynamicLink"/>

        </xsl:template>

        <!-- Generate the java managed bean files -->
        <xsl:template mode="ManagedBeanMethodsGeneration.DynamicLink"
        match="self.oclIsKindOf(uml20::classes::Dependency) and
        self.supplier.oclAsType(uml20::classes::Class).stereotypes = OrderedSet{'Managed
        Bean'}">
                <xsl:element name="ManagedBean">
                        <xsl:attribute name="Name">
                                <xsl:value-of
                                select="self.supplier.oclAsType(uml20::classes::Class).nam
                                e"/>
                        </xsl:attribute>
                <xsl:element name="Method">
                        <xsl:attribute name="Name">
                                <xsl:value-of
                                select="self.getPropertyValue('ProfileWAE_4_JSF_x__y_Dynam
                                ic_Link_x__y_Action')"/>
                        </xsl:attribute>
                   </xsl:element>
                </xsl:element>
        </xsl:template>
</xsl:stylesheet>
```

8.2 WAE4.NET a código

8.2.1 Generación vistas ASP.NET MVC

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns:xta="http://www.borland.com/xta"
          xsi:schemaLocation=
       "http://www.w3.org/1999/XSL/Transform
       platform:/plugin/com.borland.tg.xsl.core/schema/schema-for-xslt20.xsd">
    <xta:metamodel uri="http://www.borland.com/together/uml"/>
    <xta:metamodel uri="http://www.borland.com/together/uml20"/>
    <xsl:include href="ASPNetMasterPageGeneration.xsl"/>
    <xsl:variable name="ApplicationName">
    </xsl:variable>
    <xsl:variable name="ControllerName">
    </xsl:variable>
    <xsl:variable name="ViewModel">
    </xsl:variable>

        <xsl:template name="ViewsGeneration"
        match="self.oclIsKindOf(uml::together::Model)">
                <xsl:variable name="ApplicationName">
                        <xsl:value-of select="self.name"/>
                </xsl:variable>

                <xsl:element name="Folder">
                        <xsl:attribute name="Name">
                                <xsl:value-of select="'Views'"/>
                        </xsl:attribute>

                        <!-- For each package apply this rule -->
                      <xsl:apply-templates select="self.ownedMembers->select(o|
                      o.oclIsKindOf(uml::kernel::packages::Package))"
                      mode="ViewsGeneration.Packages"/>

                    <!-- Create the 'Master Page' pages -->
                    <xsl:call-template name="MasterPageGeneration" />

                </xsl:element>
    </xsl:template>

        <xsl:template mode="ViewsGeneration.Packages"
        match="self.oclIsKindOf(uml::kernel::packages::Package)">

        <!-- For each "Controller" class apply this rule -->
        <xsl:apply-templates select=
        "self.ownedMembers->select(o |o.oclIsKindOf(uml20::classes::Class) and
        o.stereotypes = OrderedSet{'Controller'})" mode="ViewsGeneration.Controller"/>

        <xsl:apply-templates select=
        "self.ownedMembers->select(o | o.oclIsKindOf(uml::kernel::packages::Package))"
        mode="ViewsGeneration.Packages"/>

  </xsl:template>

    <!-- Generate the controller folders -->
        <xsl:template mode="ViewsGeneration.Controller" match=
        "self.oclIsKindOf(uml20::classes::Class) and self.stereotypes =
        OrderedSet{'Controller'}">
                <xsl:variable name="ControllerName">
                        <xsl:value-of select="self.name"/>
                </xsl:variable>

        <xsl:element name="Folder">
                <xsl:attribute name="Name">
                        <xsl:value-of select="self.name.concat('Controller')"/>
                </xsl:attribute>

                <!--<xsl:apply-templates select="self.dependencies->select(o |
                o.stereotypes = OrderedSet{'Contains'})->collect(dc | dc.supplier)"
                mode="ViewsGeneration.Actions"/>-->

                <xsl:apply-templates select=
```

```
        "self.associations->collect(association | association.supplier)"
        mode="ViewsGeneration.Actions"/>

        </xsl:element>
</xsl:template>

<!-- For each "Action" class contained in a controller -->
<xsl:template mode="ViewsGeneration.Actions" match=
"self.oclIsKindOf(uml20::classes::Class) and self.stereotypes =
OrderedSet{'Action'}">

        <xsl:apply-templates select="self.dependencies->select(o | o.stereotypes
        = OrderedSet{'Return'})" mode="ViewsGeneration.Return"/>

</xsl:template>

<!-- For each "Return" dependency from an action -->
<xsl:template mode="ViewsGeneration.Return" match=
"self.oclIsKindOf(uml20::classes::Dependency)">
        <xsl:variable name="ViewModel">
                <xsl:value-of
                select="self.getPropertyValue('WAE_4_ASPNET_MVC_x__y_Return_x__y_V
                iewModel')"/>
        </xsl:variable>

        <xsl:apply-templates select="self.supplier" mode="ViewsGeneration.ASPX"/>

</xsl:template>

<!-- For each "ASPX" class returned by an action -->
<xsl:template mode="ViewsGeneration.ASPX"
match="self.oclIsKindOf(uml20::classes::Class) and self.stereotypes =
OrderedSet{'ASPX Server Page'}">

        <!-- For each "ASPX Server Page" class apply this rule -->
        <xsl:element name="File">
        <xsl:attribute name="Name">
                <xsl:value-of select="self.name +'.aspx'"/>
        </xsl:attribute>
        <xsl:attribute name="Type">
                <xsl:value-of select="'Content'"/>
        </xsl:attribute>

        <xsl:element name="Content">
                <xsl:element name="Page">
                        <xsl:attribute name="Title">
                                <xsl:value-of select="self.name"/>
                        </xsl:attribute>
                        <xsl:attribute name="Language">
                                <xsl:value-of select="'C#'"/>
                        </xsl:attribute>
                        <xsl:if test="self.dependencies->select(o | o.stereotypes
                        = OrderedSet{'Use'})->exists(true)">
                                <xsl:attribute name="MasterPageFile">
                                        <xsl:value-of
                                        select="'/'.concat(replace(self.dependencie
                                        s->select(o | o.stereotypes =
                                        OrderedSet{'Use'})->collect(dc |
                                        dc.supplier)-
                                        >any(true).oclAsType(uml20::classes::Class)
                                        .fullName, '.', '/'))"/>
                                </xsl:attribute>
                        </xsl:if>
                        <xsl:attribute name="Inherits">
                                <xsl:value-of
                                select="'System.Web.Mvc.ViewPage'.concat('-
                                ').concat(ApplicationName).concat('.ViewModels.').c
                                oncat(ControllerName +
                                '.').concat(ViewModel).concat('-')"/>
                        </xsl:attribute>
                        <xsl:element name="AspContent">
                                <xsl:attribute name="ID">
                                        <xsl:value-of
                                        select="self.name.concat('Content')"/>
                                </xsl:attribute>
                                <xsl:attribute name="ContentPlaceHolderID">
```

```
                                        <xsl:value-of
                                        select="self.getPropertyValue('WAE_4_ASPNET
                                        _MVC_x__y_ASPX_Server_Page_x__y_DefaultPlac
                                        eHolder')"/>
                                </xsl:attribute>
                                <xsl:attribute name="runat">
                                        <xsl:value-of select="'server'"/>
                                </xsl:attribute>

                                <xsl:if test="self.associations->select(association
                                |
                                association.supplier.oclAsType(uml20::classes::Clas
                                s).stereotypes = OrderedSet{'ASPX Form'})-
                                >exists(true)">
                                        <xsl:element name="Forms">
                                                <xsl:apply-templates
                                                select="self.associations-
                                                >collect(dc | dc.supplier)"
                                                mode="ViewsGeneration.Form"/>
                                        </xsl:element>
                                </xsl:if>

                                <xsl:if test="self.dependencies->select(o |
                                o.stereotypes = OrderedSet{'RedirectToAction'})-
                                >exists(true)">
                                        <xsl:element name="Links">
                                                <xsl:apply-templates
                                                select="self.dependencies->select(o
                                                | o.stereotypes =
                                                OrderedSet{'RedirectToAction'})-
                                                >collect(dc | dc.supplier)"
                                                mode="ViewsGeneration.ActionTo"/>
                                        </xsl:element>
                                </xsl:if>

                        </xsl:element>

                </xsl:element>
        </xsl:element>

        </xsl:element>

</xsl:template>

<!-- For the "Form" contained in a ClientPage -->
<xsl:template mode="ViewsGeneration.Form"
match="self.oclIsKindOf(uml20::classes::Class) and self.stereotypes =
OrderedSet{'ASPX Form'}">
        <xsl:variable name="ActionName">
                <xsl:value-of select="self.dependencies->select(o | o.stereotypes
                = OrderedSet{'Action Submit'})->collect(dc | dc.supplier)-
                >any(true).oclAsType(uml20::classes::Class).name"/>
        </xsl:variable>

        <xsl:element name="Form">
                <xsl:attribute name="Action">
                        <xsl:value-of select="ActionName"/>
                </xsl:attribute>
                <xsl:attribute name="Controller">
                        <xsl:value-of select="ControllerName"/>
                </xsl:attribute>
    </xsl:element>

</xsl:template>

<!-- For the "Action" ASPX page redirects to -->
<xsl:template mode="ViewsGeneration.ActionTo"
match="self.oclIsKindOf(uml20::classes::Class) and self.stereotypes =
OrderedSet{'Action'}">
        <xsl:element name="Link">
                        <xsl:attribute name="Label">
                                <xsl:value-of select="'LinkTo'.concat(self.name)"/>
                        </xsl:attribute>
                        <xsl:attribute name="Action">
                                <xsl:value-of select="self.name"/>
                        </xsl:attribute>
                        <xsl:attribute name="Controller">
                                <xsl:value-of select="ControllerName"/>
        </xsl:attribute>
```

```
            </xsl:element>
        </xsl:template>

</xsl:stylesheet>
```

8.2.2 Generación controllers ASP.NET MVC

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns:xta="http://www.borland.com/xta"
          xsi:schemaLocation="http://www.w3.org/1999/XSL/Transform
        platform:/plugin/com.borland.tg.xsl.core/schema/schema-for-xslt20.xsd">
    <xta:metamodel uri="http://www.borland.com/together/uml"/>
    <xta:metamodel uri="http://www.borland.com/together/uml20"/>
    <xsl:variable name="Namespace">
    </xsl:variable>

        <xsl:template name="ControllersGeneration"
        match="self.oclIsKindOf(uml::together::Model)">
                <xsl:variable name="Namespace">
                        <xsl:value-of select="self.name.concat('.Controllers')"/>
                </xsl:variable>

                <xsl:element name="Folder">
                        <xsl:attribute name="Name">
                                <xsl:value-of select="'Controllers'"/>
                        </xsl:attribute>

                        <!-- For each package apply this rule -->
                     <xsl:apply-templates select="self.ownedMembers->select(o|
                     o.oclIsKindOf(uml::kernel::packages::Package))"
                     mode="ControllersGeneration.Packages"/>
                </xsl:element>
    </xsl:template>

        <xsl:template mode="ControllersGeneration.Packages"
        match="self.oclIsKindOf(uml::kernel::packages::Package)">

                 For each "Controller" class apply this rule -->
                <xsl:apply-templates select="self.ownedMembers->select(o
                |o.oclIsKindOf(uml20::classes::Class) and o.stereotypes =
                OrderedSet{'Controller'})" mode="ControllersGeneration.Controller"/>
                <xsl:apply-templates select="self.ownedMembers->select(o |
                o.oclIsKindOf(uml::kernel::packages::Package))"
                mode="ControllersGeneration.Packages"/>

        </xsl:template>

        <!-- Generate the controller class files -->
        <xsl:template mode="ControllersGeneration.Controller"
        match="self.oclIsKindOf(uml20::classes::Class) and self.stereotypes =
        OrderedSet{'Controller'}">

        <xsl:element name="File">
                <xsl:attribute name="Name">
                        <xsl:value-of select="self.name.concat('Controller.cs')"/>
                </xsl:attribute>
                <xsl:attribute name="Type">
                        <xsl:value-of select="'ClassType'"/>
                </xsl:attribute>
                <xsl:attribute name="Namespace">
                        <xsl:copy-of select="'namespace '.concat(Namespace)"/>
                </xsl:attribute>
                <xsl:attribute name="Class">
                        <xsl:value-of select="self.name.concat('Controller')"/>
                </xsl:attribute>

                <xsl:element name="Actions">

                        <!--<xsl:apply-templates select="self.dependencies->select(o |
                        o.stereotypes = OrderedSet{'Contains'})->collect(dc |
                        dc.supplier)" mode="ControllersGeneration.Actions"/> -->
```

```
                        <xsl:apply-templates select="self.associations-
                        >collect(association | association.supplier)"
                        mode="ControllersGeneration.Actions"/>

                </xsl:element>

            </xsl:element>
        </xsl:template>

        <!-- Generate the "Actions" in each controller -->
        <xsl:template mode="ControllersGeneration.Actions"
        match="self.oclIsKindOf(uml20::classes::Class) and self.stereotypes =
        OrderedSet{'Action'}">

        <xsl:element name="Action">
                        <xsl:attribute name="Name">
                                <xsl:copy-of select="self.name"/>
                    </xsl:attribute>
            </xsl:element>
        </xsl:template>

</xsl:stylesheet>
```

8.2.3 Generación actions ASP.NET MVC

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns:xta="http://www.borland.com/xta"
          xsi:schemaLocation="http://www.w3.org/1999/XSL/Transform
        platform:/plugin/com.borland.tg.xsl.core/schema/schema-for-xslt20.xsd">
    <xta:metamodel uri="http://www.borland.com/together/uml"/>
    <xta:metamodel uri="http://www.borland.com/together/uml20"/>

        <xsl:template name="ActionsGeneration"
        match="self.oclIsKindOf(uml::together::Model)">

                <xsl:element name="Actions">

                <!-- For each package apply this rule -->
                <xsl:apply-templates select="self.ownedMembers->select(o|
                o.oclIsKindOf(uml::kernel::packages::Package))"
                mode="ActionsGeneration.Packages"/>

                </xsl:element>
    </xsl:template>

        <xsl:template mode="ActionsGeneration.Packages"
        match="self.oclIsKindOf(uml::kernel::packages::Package)">

                <!-- For each "Form" class apply this rule -->
                <xsl:apply-templates select="self.ownedMembers->select(o
                |o.oclIsKindOf(uml20::classes::Class) and o.stereotypes =
                OrderedSet{'ASPX Form'})" mode="ActionsGeneration.Form"/>

                <!-- For each "ASPX" class apply this rule -->
                <xsl:apply-templates select="self.ownedMembers->select(o
                |o.oclIsKindOf(uml20::classes::Class) and (o.stereotypes =
                OrderedSet{'ASPX Server Page'} or o.stereotypes =
                OrderedSet{'ASPNETMVCHandler'}))" mode="ActionsGeneration.ASPX"/>

                <!-- For each "Action" class apply this rule -->
                <xsl:apply-templates select="self.ownedMembers->select(o |
                o.oclIsKindOf(uml20::classes::Class) and o.stereotypes =
                OrderedSet{'Action'})" mode="ActionsGeneration.Action"/>

                <xsl:apply-templates select="self.ownedMembers->select(o |
                o.oclIsKindOf(uml::kernel::packages::Package))"
                mode="ActionsGeneration.Packages"/>

        </xsl:template>

        <!-- For each "Form" class apply this rule -->
        <xsl:template mode="ActionsGeneration.Form"
        match="self.oclIsKindOf(uml20::classes::Class) and self.stereotypes =
        OrderedSet{'ASPX Form'}">
```

```
         <!-- For each "Action Submit" dependency apply this rule -->
        <xsl:apply-templates select="self.dependencies->select(o | o.stereotypes
        = OrderedSet{'Action Submit'})"
        mode="ActionsGeneration.PostActionSubmit"/>

</xsl:template>

<!-- For each "ASPX" class apply this rule -->
<xsl:template mode="ActionsGeneration.ASPX"
match="self.oclIsKindOf(uml20::classes::Class) and (self.stereotypes =
OrderedSet{'ASPX Server Page'} or self.stereotypes =
OrderedSet{'ASPNETMVCHandler'})">

<!-- For each "RedirectToAction" dependency from a "ASPX" class to "Action" class
apply this rule -->
<xsl:apply-templates select="self.dependencies->select(o | o.stereotypes =
OrderedSet{'RedirectToAction'})"
mode="ActionsGeneration.PostASPXRedirectToAction"/>

</xsl:template>

<!-- For each "Action Submit" dependency apply this rule -->
<xsl:template mode="ActionsGeneration.PostActionSubmit"
match="self.oclIsKindOf(uml20::classes::Dependency) and self.stereotypes =
OrderedSet{'Action Submit'}">

        <xsl:element name="Action">
                <xsl:attribute name="Name">
                        <xsl:copy-of
                        select="self.supplier.oclAsType(uml20::classes::Class).nam
                        e"/>
                </xsl:attribute>
                <xsl:attribute name="Alias">
                        <xsl:copy-of
                        select="self.supplier.oclAsType(uml20::classes::Class).nam
                        e"/>
                </xsl:attribute>
                <xsl:attribute name="Method">
                        <xsl:copy-of select="'Post'"/>
                </xsl:attribute>
                <xsl:attribute name="InputModel">
                        <xsl:copy-of
                        select="self.getPropertyValue('WAE_4_ASPNET_MVC_x__y_Actio
                        n_Submit_x__y_InputModel')"/>
                </xsl:attribute>
                <xsl:element name="Redirects">
                        <xsl:apply-templates select="self.supplier"
                        mode="ActionsGeneration.PostAction"/>
                </xsl:element>
        </xsl:element>

</xsl:template>

<!-- For each "RedirectToAction" dependency from a "ASPX" class to "Action" class
apply this rule -->
<xsl:template mode="ActionsGeneration.PostASPXRedirectToAction"
match="self.oclIsKindOf(uml20::classes::Dependency) and self.stereotypes =
OrderedSet{'RedirectToAction'} and
self.supplier.oclAsType(uml20::classes::Class).stereotypes =
OrderedSet{'Action'}">

<xsl:element name="Action">
        <xsl:attribute name="Name">
                <xsl:copy-of
                select="self.supplier.oclAsType(uml20::classes::Class).name"/>
           </xsl:attribute>
           <xsl:attribute name="Alias">
                <xsl:copy-of
                select="self.supplier.oclAsType(uml20::classes::Class).name"/>
           </xsl:attribute>
           <xsl:attribute name="Method">
                <xsl:copy-of select="'Get'"/>
           </xsl:attribute>
           <xsl:attribute name="InputModel">
```

```
                        <xsl:copy-of
                        select="self.getPropertyValue('WAE_4_ASPNET_MVC_x__y_RedirectToAct
                        ion_x__y_InputModel')"/>
                    </xsl:attribute>
                     <xsl:element name="Redirects">
                        <xsl:apply-templates select="self.supplier"
                        mode="ActionsGeneration.PostAction"/>
                     </xsl:element>
                </xsl:element>

        </xsl:template>

      <xsl:template mode="ActionsGeneration.PostAction"
      match="self.oclIsKindOf(uml20::classes::Class) and self.stereotypes =
      OrderedSet{'Action'}">

         <xsl:apply-templates select="self.dependencies->select(o | o.stereotypes =
        OrderedSet{'RedirectToAction'})" mode="ActionsGeneration.PostRedirectToAction"/>

        <xsl:apply-templates select="self.dependencies->select(o | o.stereotypes =
        OrderedSet{'Return'})" mode="ActionsGeneration.PostReturn"/>

</xsl:template>

<xsl:template mode="ActionsGeneration.PostRedirectToAction"
match="self.oclIsKindOf(uml20::classes::Dependency) and self.stereotypes =
OrderedSet{'RedirectToAction'}">

        <!-- Create RedirectToAction node -->
        <xsl:element name="RedirectToAction">
                    <xsl:attribute name="Action">
                            <xsl:copy-of
                            select="self.supplier.oclAsType(uml20::classes::Class).nam
                            e.concat('ViaGet')"/>
                    </xsl:attribute>
                </xsl:element>

        </xsl:template>

                <xsl:template mode="ActionsGeneration.PostReturn"
                match="self.oclIsKindOf(uml20::classes::Dependency) and self.stereotypes
                = OrderedSet{'Return'}">

        <!-- Create RedirectToAction node -->
        <xsl:element name="RedirectToAction">
                    <xsl:attribute name="Action">
                            <xsl:copy-of
                            select="self.client.oclAsType(uml20::classes::Class).name.
                            concat(self.supplier.oclAsType(uml20::classes::Class).name
                            ).concat('ViaGet')"/>
                    </xsl:attribute>
                </xsl:element>

        </xsl:template>

        <!-- For each "Action" class apply this rule -->
        <xsl:template mode="ActionsGeneration.Action"
        match="self.oclIsKindOf(uml20::classes::Class) and self.stereotypes =
        OrderedSet{'Action'}">

                <!-- For each "RedirectToAction" dependency from an "Action" class to an
                "Action" class apply this rule -->
                <xsl:apply-templates select="self.dependencies->select(o | o.stereotypes
                = OrderedSet{'RedirectToAction'} and
                o.supplier.oclAsType(uml20::classes::Class).stereotypes =
                OrderedSet{'Action'})" mode="ActionsGeneration.RedirectToAction"/>

                <!-- For each "Return" dependency from an "Action" class to an "ASPX"
                class apply this rule -->
                <xsl:apply-templates select="self.dependencies->select(o | o.stereotypes
                = OrderedSet{'Return'} and
                o.supplier.oclAsType(uml20::classes::Class).stereotypes =
                OrderedSet{'ASPX Server Page'})" mode="ActionsGeneration.Return"/>

        </xsl:template>

        <!-- For each "RedirectToAction" dependency from an "Action" class to an "Action"
        class apply this rule -->
```

```
    <xsl:template mode="ActionsGeneration.RedirectToAction"
    match="self.oclIsKindOf(uml20::classes::Dependency) and self.stereotypes =
    OrderedSet{'RedirectToAction'} and
    self.client.oclAsType(uml20::classes::Class).stereotypes = OrderedSet{'Action'} and
    self.supplier.oclAsType(uml20::classes::Class).stereotypes = OrderedSet{'Action'}">

      <xsl:element name="Action">
            <xsl:attribute name="Name">
                  <xsl:copy-of
                  select="self.supplier.oclAsType(uml20::classes::Class).name.concat
                  ('ViaGet')"/>
            </xsl:attribute>
            <xsl:attribute name="Alias">
                  <xsl:copy-of
                  select="self.supplier.oclAsType(uml20::classes::Class).name.concat
                  ('ViaGet')"/>
            </xsl:attribute>
            <xsl:attribute name="Method">
                  <xsl:copy-of select="'Get'"/>
            </xsl:attribute>
            <xsl:attribute name="InputModel">
                  <xsl:copy-of
                  select="self.getPropertyValue('WAE_4_ASPNET_MVC_x__y_RedirectToAct
                  ion_x__y_InputModel')"/>
            </xsl:attribute>
            <xsl:element name="Redirects">
                  <xsl:apply-templates select="self.supplier"
                  mode="ActionsGeneration.PostAction"/>
               </xsl:element>
            </xsl:element>

      </xsl:template>

      <!-- For each "Return" dependency from an "Action" class to an "ASPX" class apply
      this rule -->
      <xsl:template mode="ActionsGeneration.Return"
      match="self.oclIsKindOf(uml20::classes::Dependency) and self.stereotypes =
      OrderedSet{'Return'} and self.client.oclAsType(uml20::classes::Class).stereotypes
      = OrderedSet{'Action'} and
      self.supplier.oclAsType(uml20::classes::Class).stereotypes = OrderedSet{'ASPX
      Server Page'}">

      <xsl:element name="Action">
            <xsl:attribute name="Name">
                  <xsl:copy-of
                  select="self.client.oclAsType(uml20::classes::Class).name.concat(s
                  elf.supplier.oclAsType(uml20::classes::Class).name).concat('ViaGet
                  ')"/>
               </xsl:attribute>
               <xsl:attribute name="Alias">
                  <xsl:copy-of
                  select="self.client.oclAsType(uml20::classes::Class).name.concat(s
                  elf.supplier.oclAsType(uml20::classes::Class).name).concat('ViaGet
                  ')"/>
               </xsl:attribute>
               <xsl:attribute name="Method">
                  <xsl:copy-of select="'Get'"/>
               </xsl:attribute>
               <xsl:attribute name="ViewModel">
                  <xsl:copy-of
                  select="self.getPropertyValue('WAE_4_ASPNET_MVC_x__y_Return_x__y_V
                  iewModel')"/>
               </xsl:attribute>
               <xsl:attribute name="ToView">
                  <xsl:copy-of
                  select="self.supplier.oclAsType(uml20::classes::Class).name"/>
               </xsl:attribute>
            </xsl:element>

      </xsl:template>

</xsl:stylesheet>
```

8.3 E-WAE a WAE4JSF

8.3.1 Paquete E-WAE a paquete WAE4JSF

```
 transformation AIWAEModel_To_WAE4JSFModel;

import AIWAEPackage_To_WAE4JSFPackage;
import AIWAEDependency_To_WAE4JSFDependency;
import AIWAEAssociation_To_WAE4JSFAssociation;
import AIWAETaggedValues_To_WAE4JSFTaggedValues;

metamodel 'http://www.borland.com/together/uml';
metamodel 'http://www.borland.com/together/uml20';

-- MODEL TO MODEL : ENTRY TRANSFORMATION POINT

mapping main(in AIWAEModel: uml::together::Model): uml::together::Model
{
    init
    {
            -- PACKAGES

            var AIWAEPackages := AIWAEModel.nestedPackages;

            var WAE4JSFPackages := AIWAEPackages->collect(AIWAEPackage |
            AIWAEPackage.ToWAE4JSFPackage());

            -- DEPENDENCIES

            var AIWAEDependencies :=
            AIWAEModel.allInstances(uml20::classes::Dependency);

            var WAE4JSFDependencies := AIWAEDependencies
                        -- MAP EACH AI WAE DEPENDENCY TO WAE4JSF DEPENDENCY
                        ->collect(AIWAEDependency |
                        AIWAEDependency.ToWAE4JSFDependency().CopyDependencyTagged
                        Values())
                        -- CREATES WAE4JSF HIDDEN DEPENDENCIES FROM AI WAE MODEL
                        ->union(AIWAEDependencies->collect(AIWAEDependency |
                        AIWAEDependency.GetWAE4JSFHiddenDependencies(AIWAEModel)))
                        ;

            -- CLASS ASSOCIATIONS

            var AIWAEClasses := AIWAEModel.allInstances(uml20::classes::Class)-
            >collect(AIWAEClass | AIWAEClass.SetAssociations());

            -- MAP DATAFIELD ASSOCIATIONS TO TAGGED VALUES

            var AIWAEDataFieldDependencies :=
            AIWAEModel.allInstances(uml20::kernel::KernelAssociation)
                                        ->select(a |
                                        a.supplier.oclAsType(uml20::classes::Cl
                                        ass).stereotypes =
                                        OrderedSet{'DataCarrier'})
                                ->collect(a | a.SetDataFieldsToTaggedValues())
                                }
    object
    {
            nestedPackages := WAE4JSFPackages->asOrderedSet();
    }

}
```

8.3.2 Asociación E-WAE a asociación WAE4JSF

```
transformation AIWAEAssociation_To_WAE4JSFAssociation;

metamodel 'http://www.borland.com/together/uml';
metamodel 'http://www.borland.com/together/uml20';

-- ASSOCIATIONS

mapping uml20::classes::Class::SetAssociations()
```

```
{
    init
    {
            var Associations := self.associations->collect(a |
            a.ToWAE4JSFServerPageJSFFormAssociation())
                                                    ->union(self.associations-
            >collect(a | a.ToWAE4JSFServerPageJSFDivAssociation()))
                                                    ->union(self.associations-
            >collect(a | a.ToWAE4JSFDivJSFDivAssociation()))
                                                    ->union(self.associations-
            >collect(a | a.ToWAE4JSFDivJSFFormAssociation()));
    }
}

-- MAP COMPOSITION ASSOCIATIONS BETWEEN JSFSERVERPAGE-JSFFORM
mapping uml20::kernel::KernelAssociation::ToWAE4JSFServerPageJSFFormAssociation() :
uml20::kernel::KernelAssociation
when
{
    self.client.stereotypes = OrderedSet{'Page'}
    and
    self.supplier.stereotypes = OrderedSet{'Form'}
}
{
    init
    {
            var WAE4JSFClient   := self.client.resolve(uml20::classes::Class)->any(c
            | c.stereotypes = OrderedSet{'JSF Server Page'});

            var WAE4JSFSupplier := self.supplier.resolve(uml20::classes::Class)-
            >any(c | c.stereotypes = OrderedSet{'JSF Form'});
    }
    object
    {
            client := WAE4JSFClient->any(true);

            supplier := WAE4JSFSupplier->any(true);

            associatesType := self.associatesType;
    }
}

-- MAP COMPOSITION ASSOCIATIONS BETWEEN JSFSERVERPAGE-JSFDIV
mapping uml20::kernel::KernelAssociation::ToWAE4JSFServerPageJSFDivAssociation() :
uml20::kernel::KernelAssociation
when
{
    self.client.stereotypes = OrderedSet{'Page'}
    and
    self.supplier.stereotypes = OrderedSet{'Div'}
}
{
    init
    {
            var WAE4JSFClient   := self.client.resolve(uml20::classes::Class)->any(c
            | c.stereotypes = OrderedSet{'JSF Server Page'});

            var WAE4JSFSupplier := self.supplier.resolve(uml20::classes::Class)-
            >any(c | c.stereotypes = OrderedSet{'JSF Div'});
    }
    object
    {
            client := WAE4JSFClient->any(true);

            supplier := WAE4JSFSupplier->any(true);

            associatesType := self.associatesType;
    }
}

-- MAP COMPOSITION ASSOCIATIONS BETWEEN DIVs
mapping uml20::kernel::KernelAssociation::ToWAE4JSFDivJSFDivAssociation() :
uml20::kernel::KernelAssociation
when
{
    self.client.stereotypes = OrderedSet{'Div'}
    and
```

```
    self.supplier.stereotypes = OrderedSet{'Div'}
}
{
    init
    {
            var WAE4JSFClient   := self.client.resolve(uml20::classes::Class)->any(c
            | c.stereotypes = OrderedSet{'JSF Div'});

            var WAE4JSFSupplier := self.supplier.resolve(uml20::classes::Class)-
            >any(c | c.stereotypes = OrderedSet{'JSF Div'});
    }
    object
    {
            client := WAE4JSFClient->any(true);

            supplier := WAE4JSFSupplier->any(true);

            associatesType := self.associatesType;
    }
}

-- MAP COMPOSITION ASSOCIATIONS BETWEEN JSFDIV-JSFFORM
mapping uml20::kernel::KernelAssociation::ToWAE4JSFDivJSFFormAssociation() :
uml20::kernel::KernelAssociation
when
{
    self.client.stereotypes = OrderedSet{'Div'}
    and
    self.supplier.stereotypes = OrderedSet{'Form'}
}
{
    init
    {
            var WAE4JSFClient   := self.client.resolve(uml20::classes::Class)->any(c
            | c.stereotypes = OrderedSet{'JSF Div'});

            var WAE4JSFSupplier := self.supplier.resolve(uml20::classes::Class)-
            >any(c | c.stereotypes = OrderedSet{'JSF Form'});
    }
    object
    {
            client := WAE4JSFClient->any(true);

            supplier := WAE4JSFSupplier->any(true);

            associatesType := self.associatesType;
    }
}
```

8.3.3 Clase E-WAE a dependencia WAE4JSF

```
transformation AIWAEClass_To_WAE4JSFDependency;

metamodel 'http://www.borland.com/together/uml';
metamodel 'http://www.borland.com/together/uml20';

mapping uml20::classes::Class::ToJSFDynamicLink(in AIWAELinkDependencyToFunction :
uml20::classes::Dependency) : uml20::classes::Dependency
{
    init
    {
            var AIWAEClient :=
            AIWAELinkDependencyToFunction.client.oclAsType(uml20::classes::Class);

            var WAE4JSFClient := AIWAEClient.resolve(uml20::classes::Class)->any(c |
            c.stereotypes = OrderedSet{'JSF Server Page'});

            var TaggedValue :=
            AIWAELinkDependencyToFunction.supplier.oclAsType(uml20::classes::Class).n
            ame;
    }
    object
    {
            client := WAE4JSFClient;

            supplier := self;
```

```
            stereotypes := OrderedSet{'Dynamic Link'};

            description := TaggedValue;
    }
}

mapping uml20::classes::Class::ToJSFDynamicLinkFromTemplateComponentDependency(in
AIWAELinkDependencyToFunction : uml20::classes::Dependency) :
uml20::classes::Dependency
{
    init
    {
            var AIWAEClient :=
            AIWAELinkDependencyToFunction.client.oclAsType(uml20::classes::Class);

            var WAE4JSFClient := AIWAEClient.resolve(uml20::classes::Class)->any(c |
            c.stereotypes = OrderedSet{'TCData'});

            var TaggedValue :=
            AIWAELinkDependencyToFunction.client.oclAsType(uml20::classes::Class).nam
            e;
    }
    object
    {
            client := WAE4JSFClient;

            supplier := self;

            stereotypes := OrderedSet{'Dynamic Link'};

            description := TaggedValue;
    }
}

mapping uml20::classes::Class::ToJSFDynamicLinkFromDiv(in
AIWAELinkDependencyToFunction : uml20::classes::Dependency) :
uml20::classes::Dependency
{
    init
    {
            var AIWAEClient :=
            AIWAELinkDependencyToFunction.client.oclAsType(uml20::classes::Class);

            var WAE4JSFClient := AIWAEClient.resolve(uml20::classes::Class)->any(c |
            c.stereotypes = OrderedSet{'JSF Div'});

            var TaggedValue :=
            AIWAELinkDependencyToFunction.supplier.oclAsType(uml20::classes::Class).n
            ame;
    }
    object
    {
            client := WAE4JSFClient;

            supplier := self;

            stereotypes := OrderedSet{'Dynamic Link'};

            description := TaggedValue;
    }
}

mapping uml20::classes::Class::ToJSFDynamicSubmit(in AIWAESubmitDependencyToFunction
: uml20::classes::Dependency) : uml20::classes::Dependency
{
    init
    {
            var AIWAEClient :=
            AIWAESubmitDependencyToFunction.client.oclAsType(uml20::classes::Class);

            var WAE4JSFClient := AIWAEClient.resolve(uml20::classes::Class)->any(c |
            c.stereotypes = OrderedSet{'JSF Form'});

            var TaggedValue :=
            AIWAESubmitDependencyToFunction.supplier.oclAsType(uml20::classes::Class)
            .name;
```

```
    }
    object
    {
            client := WAE4JSFClient;

            supplier := self;

            stereotypes := OrderedSet{'Dynamic Submit'};

            description := TaggedValue;
    }
}

mapping uml20::classes::Class::ToJSFReturn(in
AIWAEReturnDependencyFromFunctionToServerPage : uml20::classes::Dependency) :
uml20::classes::Dependency
{
    init
    {
            var AIWAESupplier :=
            AIWAEReturnDependencyFromFunctionToServerPage.supplier.oclAsType(uml20::c
            lasses::Class);

            var WAE4JSFOutcomeName :=
            AIWAEReturnDependencyFromFunctionToServerPage.client.oclAsType(uml20::cla
            sses::Class).name + ' To ' +

            AIWAEReturnDependencyFromFunctionToServerPage.supplier.oclAsType(uml20::c
            lasses::Class).name;

            var WAE4JSFSupplier := AIWAESupplier.resolve(uml20::classes::Class)-
            >any(c | c.stereotypes = OrderedSet{'Outcome'} and c.name =
            WAE4JSFOutcomeName);

            var TaggedValue :=
            AIWAEReturnDependencyFromFunctionToServerPage.client.oclAsType(uml20::cla
            sses::Class).name;
    }
    object
    {
            client := self;

            supplier := WAE4JSFSupplier;

            stereotypes := OrderedSet{'Return'};

            description := TaggedValue;
    }
}

mapping uml20::classes::Class::ToDtoDependecy(in AIWAEDependencyToFunction :
uml20::classes::Dependency) : uml20::classes::Dependency
{
    init
    {
            var AIWAEClient :=
            AIWAEDependencyToFunction.client.oclAsType(uml20::classes::Class);

            var AIWAEAssociation := AIWAEClient.associations->select(a |
            a.supplier.stereotypes = OrderedSet{'DataCarrier'})->any(true);

            var WAE4JSFSupplier :=
            AIWAEAssociation.supplier.oclAsType(uml20::classes::Class).resolve(uml20:
            :classes::Class)->any(c | c.stereotypes = OrderedSet{'DTO'})

    }
    object
    {
            client := self;

            supplier := WAE4JSFSupplier;
    }
}
```

8.4 E-WAE a WAE4.NET

8.4.1 Paquete E-WAE a paquete WAE4.NET

```
transformation AIWAEPackage_To_WAE4NETPackage;

import AIWAEClass_To_WAE4NETClass;

metamodel 'http://www.borland.com/together/uml';
metamodel 'http://www.borland.com/together/uml20';

-- PACKAGE TO PACKAGE

mapping uml::kernel::packages::Package::ToWAE4NETPackage() :
uml::kernel::packages::Package
{
	init
	{
		-- CLASSES

		var AIWAEClasses := self.ownedMembers->select(o |
		o.oclIsKindOf(uml20::classes::Class));

		var WAE4NETClasses := AIWAEClasses
				-- MAP EACH AI WAE CLASS TO WAE4NET CLASS
				 ->collect(AIWAEClass |
				 AIWAEClass.oclAsType(uml20::classes::Class).ToWAE4NETClass
				 ())
				-- CREATES WAE4NET HIDDEN CLASSES THE FROM AI WAE MODEL
				 ->union(AIWAEClasses->collect(AIWAEClass |
				 AIWAEClass.oclAsType(uml20::classes::Class).GetWAE4NETHidd
				 enClasses()));

		-- NESTED PACKAGE

		var AIWAENestedPackages := self.nestedPackages;

		var WAE4NETNestedPackages := AIWAENestedPackages->collect(AIWAEPackage |
		AIWAEPackage.ToWAE4NETPackage());
	}
	object
	{
		name := self.name;

		nestedPackages += WAE4NETNestedPackages->asOrderedSet();

		ownedMembers += WAE4NETClasses->asOrderedSet();
	}
}
```

8.4.2 Asociación E-WAE a asociación WAE.NET

```
transformation AIWAEAssociation_To_WAE4NETAssociation;

metamodel 'http://www.borland.com/together/uml';
metamodel 'http://www.borland.com/together/uml20';

-- ASSOCIATIONS

mapping uml20::classes::Class::SetAssociations()
{
	init
	{
		var Associations := self.associations->
					collect(o | o.ToWAE4NETAssociation())
					                                -
					>union(self.GetWAE4NETLinkAssociation())
					                                -
					>union(self.GetWAE4NETSubmitAssociation());
```

```
	}
}

-- CREATES COMPOSITION ASSOCIATIONS BETWEEN ASPXSERVERPAGE-ASPXFORM AND CONTROLLER-
ACTION
query uml20::kernel::KernelAssociation::ToWAE4NETAssociation() :
Set(uml20::kernel::KernelAssociation)
{
	Set{self.ToWAE4NETASPXServerPageAssociation(),
	self.ToWAE4NETControllerAssociation()}

}

-- MAP COMPOSITION ASSOCIATIONS BETWEEN ASPXSERVERPAGE-ASPXFORM
mapping uml20::kernel::KernelAssociation::ToWAE4NETASPXServerPageAssociation() :
uml20::kernel::KernelAssociation
when
{
	self.client.stereotypes = OrderedSet{'Page'}
	and
	self.supplier.stereotypes = OrderedSet{'Form'}
}
{
	init
	{
		var WAE4NetClient   := self.client.resolve(uml20::classes::Class)->any(c
		| c.stereotypes = OrderedSet{'ASPX Server Page'});

		var WAE4NetSupplier := self.supplier.resolve(uml20::classes::Class)-
		>any(c | c.stereotypes = OrderedSet{'ASPX Form'});
	}
	object
	{
		client := WAE4NetClient->any(true);

		supplier := WAE4NetSupplier->any(true);

		associatesType := self.associatesType;
	}
}

-- CREATES COMPOSITION ASSOCIATIONS BETWEEN CONTROLLER-ACTION
mapping uml20::kernel::KernelAssociation::ToWAE4NETControllerAssociation() :
uml20::kernel::KernelAssociation
when
{
	self.client.stereotypes = OrderedSet{'Command'}
	and
	self.supplier.stereotypes = OrderedSet{'Function'}
}
{
	init
	{
		var WAE4NetClient   := self.client.resolve(uml20::classes::Class)->any(c
		| c.stereotypes = OrderedSet{'Controller'});

		var WAE4NetSupplier := self.supplier.resolve(uml20::classes::Class)-
		>any(c | c.stereotypes = OrderedSet{'Action'});
	}
	object
	{
		client := WAE4NetClient->any(true);

		supplier := WAE4NetSupplier->any(true);

		associatesType := self.associatesType;
	}
}

-- CREATES WAE4NET ASSOCIATIONS FOR EACH AI WAE LINK DEPENDENCY BETWEEN SERVER PAGES OR
TEMPLATE COMPONENT TO SERVER PAGE
query uml20::classes::Class::GetWAE4NETLinkAssociation() :
Set(uml20::kernel::KernelAssociation)
{
	self.dependencies->select(d | d.stereotypes = OrderedSet{'Link'}
	                              and                                (
	                              d.client.oclAsType(uml20::classes::Class).stereotyp
	                              es = OrderedSet{'Page'}
```

```
                                        or
                                d.client.oclAsType(uml20::classes::Class).stereotyp
                                es = OrderedSet{'TCData'}
                                )
                                                                and
                                d.supplier.oclAsType(uml20::classes::Class).stereot
                                ypes = OrderedSet{'Page'})
                        ->collect(d |
                        d.oclAsType(uml20::classes::Dependency).ToWAE4NETLinkAssoci
                        ation())->asOrderedSet()
}

-- MAP A AI WAE LINK DEPENDENCY BETWEEN SERVER PAGES OR TEMPLATE COMPONENT TO SERVER
PAGE TO ASSOCIATION BETWEEN CONTROLLER AND ACTION GENERATED CLASSES
mapping uml20::classes::Dependency::ToWAE4NETLinkAssociation() :
uml20::kernel::KernelAssociation
{
        init
        {
                var AIWAEClientName := self.client.oclAsType(uml20::classes::Class).name;

                var AIWAESupplierName :=
                self.supplier.oclAsType(uml20::classes::Class).name;

                var WAE4NETActionName := AIWAEClientName + ' To ' + AIWAESupplierName;
        }
        object
        {
                client :=
                self.client.oclAsType(uml20::classes::Class).resolve(uml20::classes::Clas
                s)->select(c | c.stereotypes = OrderedSet{'Controller'})->any(true);

                supplier :=
                self.supplier.oclAsType(uml20::classes::Class).resolve(uml20::classes::Cl
                ass)->select(c | c.stereotypes = OrderedSet{'Action'})->any(c | c.name =
                WAE4NETActionName);

                associatesType := uml20::kernel::AssociatesTypeKind::COMPOSITION;
        }
}

-- CREATES WAE4NET ASSOCIATIONS FOR EACH AI WAE SUBMIT DEPENDENCY FROM A FORM TO A
SERVER PAGE
query uml20::classes::Class::GetWAE4NETSubmitAssociation() :
Set(uml20::kernel::KernelAssociation)
{
        self.associations->select(a | a.associatesType =
        uml20::kernel::AssociatesTypeKind::COMPOSITION and
                                        a.supplier.stereotypes =
                                OrderedSet{'Form'})
        ->collect(a | a.supplier.oclAsType(uml20::classes::Class).dependencies
        ->select(d | d.stereotypes = OrderedSet{'Submit'})
        ->collect(d |
        d.oclAsType(uml20::classes::Dependency).ToWAE4NETSubmitAssociation(self)))-
        >asOrderedSet();
}

-- MAP A AI WAE SUBMIT DEPENDENCY FROM A FORM TO SERVER PAGE TO ASSOCIATION BETWEEN
CONTROLLER AND ACTION GENERATED CLASSES
mapping uml20::classes::Dependency::ToWAE4NETSubmitAssociation(in containerClass :
uml20::classes::Class) : uml20::kernel::KernelAssociation
when
{
        containerClass.stereotypes = OrderedSet{'Page'}
}
{
        init
        {
                var AIWAEClientName := self.client.oclAsType(uml20::classes::Class).name;

                var AIWAESupplierName :=
                self.supplier.oclAsType(uml20::classes::Class).name;

                var WAE4NETActionName := AIWAEClientName + ' To ' + AIWAESupplierName;
        }
        object
        {
```

```
                client := containerClass.resolve(uml20::classes::Class)->select(c |
                c.stereotypes = OrderedSet{'Controller'})->any(true);

                supplier :=
                self.supplier.oclAsType(uml20::classes::Class).resolve(uml20::classes::Cl
                ass)->select(c | c.stereotypes = OrderedSet{'Action'})->any(c | c.name =
                WAE4NETActionName);

                associatesType := uml20::kernel::AssociatesTypeKind::COMPOSITION;
        }
}
```

8.4.3 Clase E-WAE a dependencia WAE4.NET

```
transformation AIWAEClass_To_WAE4NETDependency;

metamodel 'http://www.borland.com/together/uml';
metamodel 'http://www.borland.com/together/uml20';

mapping uml20::classes::Class::ToDtoDependecy(in AIWAEDependencyToFunction :
uml20::classes::Dependency) : uml20::classes::Dependency
{
        init
        {
                var AIWAEClient :=
                AIWAEDependencyToFunction.client.oclAsType(uml20::classes::Class);

                var AIWAEAssociation := AIWAEClient.associations->select(a |
                a.supplier.stereotypes = OrderedSet{'DataCarrier'})->any(true);

                var WAE4JSFSupplier :=
                AIWAEAssociation.supplier.oclAsType(uml20::classes::Class).resolve(uml20:
                :classes::Class)->any(c | c.stereotypes = OrderedSet{'DTO'})

        }
        object
        {
                client := self;

                supplier := WAE4JSFSupplier;
        }
}
```