**CAPÍTULO 3**

**ENFOQUES METODOLÓGICOS *MDD* PARA LAS *RIA***

En el capítulo anterior se presentó una visión general de las *RIA* con sus características principales y las diferentes tecnologías utilizadas para el desarrollo de las mismas. Se han visto también, las distintas formas en que se implementan este tipo de aplicaciones, como las implementaciones basadas en librerías *Javascript*, las implementaciones basadas en la instalación de *plug-ins* en el navegador, y las basadas en ambientes en tiempo de ejecución. Las implementaciones basadas en librerías *Javascript* son las que presentan el mayor grado de estandarización, he allí que resulta la opción más popular en la comunidad *Web*.

En este capítulo se verá el enfoque de desarrollo de aplicaciones *Web* basado en modelos, presentando primeramente los conceptos de *MDD* (*Model Driven Software Engineering*), *MDD* (*Model Driven Development*) y *MDA* (*Model Driven Architecture*), para posteriormente presentar a las metodologías *Web* existentes basadas en modelos que presentan características de las *RIA*. Finalmente se presentará la metodología *Web* *MoWebA* (*Model Oriented Web Approach*), una metodología *Web* separada en capas que sigue el paradigma *MDA* para el ciclo de desarrollo de sus aplicaciones y que resulta prometedora para la implementación de características de las *RIA*.

**3.1 Model driven software engineering (*MDD*)**

Los modelos son de suma importancia para entender y compartir conocimiento acerca de un software complejo. *MDD* es concebida como una herramienta para convertir este hecho en una manera concreta de trabajar y pensar, transformando los modelos en elementos fundamentales para todo el ciclo de desarrollo en la ingeniería de software []. En *MDD*, los conceptos principales son los modelos y las transformaciones (esto es, manipulaciones y/o operaciones sobre los modelos).

*MDD* tiene como objetivo llevar a cabo el desarrollo de artefactos de software utilizando a los modelos y a las transformaciones sobre estos, como piezas clave para el logro de tal objetivo. Hoy en día se ha dado un valor extra a los modelos, debido a que no solamente sirven para mantener una mejor comunicación entre los desarrolladores y las partes interesadas en un sistema en particular (*stakeholders*), o bien para mantener los sistemas debidamente documentados, sino también, estos modelos pueden contener la suficiente expresividad y riqueza como para representar información que posteriormente puede transformase y obtener así el software deseado.

Un concepto clave en el contexto *MDD* es el de metamodelo. Con el metamodelo es posible definir la sintaxis abstracta de un lenguaje de modelado. Análogamente a las gramáticas que sirven para definir a un lenguaje de programación, el metamodelo permite representar a todos los modelos posibles que forman parte del lenguaje de modelado.

**3.1.1 *Model Driven Development* (*MDD*) *y Model Driven Architecture* (*MDA*)**

En *MDD*, es posible adoptar un enfoque *MDD* para el ciclo de desarrollo de una aplicación. *MDD* es un paradigma de desarrollo que utiliza a los modelos como artefactos primarios en el proceso de desarrollo. Usualmente en *MDD* la implementación es generada de manera automática o semiautomática a partir de los modelos. Por otra parte, *MDA*[[1]](#footnote-1) es un estándar, impulsado por el consorcio *OMG* (*Object Management Group*), que contiene en sí mismo a varios estándares de facto tales como *UML*[[2]](#footnote-2) *(Unified Modeling Language*), *OCL*[[3]](#footnote-3) (*Object Constraint Language*), *MOF*[[4]](#footnote-4) (*Meta Object Facility*), *QVT[[5]](#footnote-5)* (*Query View Transformation*), entre otros. *MDA* promueve el desarrollo de software para diversos dominios de aplicación, como las aplicaciones para el ámbito de las finanzas, las telecomunicaciones, las aplicaciones aeroespaciales, las embebidas, etc. *MDA* es un subconjunto de *MDD* que propone estándares para cada paso en el proceso de desarrollo de las aplicaciones. Utiliza un esquema de arquitectura dividida en capas como puede apreciarse en la . Los meta-metamodelos (M3) se expresan por medio de *MOF* o *ECORE* para el *Eclipse Modelling Framework* (*EMF*). Los metamodelos (M2) de la aplicación se expresan por medio de un *General Purpose Modelling Lenguage (GPML)* (por lo general *UML*) que cuenta con diversos modelos para representar los comportamientos (estáticos y dinámicos) de una aplicación en particular. La capa M2 describe los conceptos utilizados en M1 para la definición de los modelos. Finalmente el objeto del mundo real, en el ejemplo de la Figura 1 un video, se representa en M0.



Figura 1 Arquitectura dividida en capas de MDA: Objetos del mundo real (M0), modelos (M1), metamodelos (M2) y meta-metamodelos (M3)

Las fases de desarrollo con el enfoque *MDA* se presentan en la . En la primera fase se tiene el *Computation Independent Model* (*CIM*), que corresponde a los documentos, modelos o diagramas utilizados para la toma de requerimientos de una aplicación en particular, independientemente de cómo planean ser implementados. Representan el punto de vista del negocio de la solución.



Figura 2 Cadena de transformaciones en *MDA*

Los *CIM* son los puntos de entrada de los *Platform Independent Model* (*PIM*). La transformación *CIM* a *PIM* se da por lo general por medio de un mapeo manual. La fase del *PIM* contempla la representación del sistema por medio de modelos que son independientes de la tecnología de implementación. Los *PIM* pueden ser transformados a un *Platform Specific Model* (*PSM*) a través de una transformación modelo a modelo (M2M), y en muchos casos soportados por lenguajes de transformación como *QVT* o *ATL*(*Atlas Transformation Languages*). Los *PSM* son modelos enriquecidos con detalles de una plataforma destino en particular. Finalmente estos *PSM* pueden ser transformados a código fuente por medio de una transformación de modelo a texto (*M2T*), apoyándose con herramientas de trasformación *M2T* como *MOFScript*, *Acceleo* u *JET* (*Java Emmitter Template*).

**3.1.2 Conceptos básicos de la generación de código a partir de los modelos**

Uno de los aportes de *MDD* es obtener sistemas a partir de los modelos. Las plataformas de ejecución actuales son a menudo basadas en código, con pocas excepciones que permiten una interpretación directa de los modelos. De esta forma, las transformaciones *M2T* en el área *MDD* son a menudo relacionadas con la generación de código para alcanzar la transición a partir del nivel del modelo al nivel de código.

Mientras que en el contexto de los compiladores, la generación de código es el proceso de transformar el código fuente en código máquina, en el mundo *MDE* (*Model Driven Engeneering*), la generación de código es el proceso de transformar modelos en código fuente.

Entre las preguntas esenciales que se tienen que tener en cuenta si se va a desarrollar un generador de código basado en modelos, se encuentran las siguientes:

**¿Qué tanto código va a generarse?**

La pregunta principal aquí es qué parte del código puede ser automáticamente generada a partir de los modelos. ¿Es posible llevar a cabo una generación de código parcial o total? La generación parcial de código puede implicar muchas cosas en este contexto. Primero, puede implicar que una capa (horizontal o vertical) de la aplicación sea completamente generada, mientras que otra capa podría ser desarrollada completamente de manera manual. Más aún, una capa puede ser generada parcialmente y otras partes no cubiertas tienen que ser completadas con código manual. La generación parcial de código también puede referirse al nivel de modelado, utilizando solamente la generación de código para ciertas partes del modelo, mientras que otras partes no son manipuladas por el generador de código y tienen que ser implementadas manualmente.

**¿Qué código va a generarse?**

Implica qué clase de código fuente va a generarse. Por supuesto, el código a ser generado debe ser lo más conciso posible y debe ser código que puede ser entendido por los desarrolladores. La idea es generar la menor cantidad de código que sea capaz de representar un sistema de la mejor manera.

**¿Cómo va a generarse?**

Muchos lenguajes pueden ser empleados para generar código a partir de los modelos y pueden ser *GPL*(*General Purpose Languages*) y *DSL*(*Domanin Specific Languages*). Actualmente existen varios lenguajes basados en plantillas para generar texto a partir de modelos, entre los que se puede citar a *XSLT*, *JET*, *Xpand*, *MOFScript* y *Acceleo*.

**3.1.3 Una vista de los lenguajes de transformación basados en plantillas**

Diferentes lenguajes basados en *plantillas* existen en la actualidad,  los cuales pueden ser empleados para generar texto a partir de los modelos.

***XSLT*[[6]](#footnote-6)**

La serialización *XMI*[[7]](#footnote-7) de los modelos pueden ser procesados con *XSLT*, que es el estándar *W3C* para transformar documentos *XML* en documentos arbitrarios de texto. Sin embargo, en este caso, los  scripts de generación de código tienen que ser implementados basados en la serialización *XMI*, lo cual requiere ciertos conocimientos adicionales acerca de cómo los modelos son actualmente codificados como archivos *XML*. Así, el enfoque opera directamente a nivel de modelo.

***JET* - *Java* Emitter *Template*[[8]](#footnote-8)**

Fue uno de los primeros enfoques de desarrollo del *EMF* para la generación de código a partir de modelos. Pero *JET* no está limitada a modelos basados en *EMF*. En general, con *JET*, todo objeto basado en *Java* es transformable a texto. *JET* provee una sintaxis similar a *JSP* adaptada a la estructura *template* para transformación *M2T*. Expresiones *Java* arbitrarias pueden ser introducidas en los *plantillas JET*. Los *template* de *JET* son transformados a código *Java* puro para propósitos de ejecución. Sin embargo, no tiene un lenguaje de consulta dedicado para los modelos disponibles en *JET*.

**Xpand[[9]](#footnote-9)**

Este lenguaje de transformación provee un lenguaje dedicado para consultar modelos siendo este una combinación de *Java* y *OCL* (muchos iteradores basados en *OCL* están disponibles). La continuación de este proyecto se llama *Xtend*, que está basado en *Java*,y ofrece muchas características adicionales propias del lenguaje. Por ejemplo, es posible incrustar *plantillas* de generación de código (para tener una sintaxis similar  al *template* Xpand) dentro del código *Xtend*.

**MOFScript[[10]](#footnote-10)**

Este proyecto provee otro lenguaje de transformación *M2T* proveyendo características similares tales a *Xpand*. *MOFScript* ha sido desarrollado como una propuesta de estandarización para la *OMG*, se encuentra disponible como un *plug-in*para el *Eclipse* y soporta modelos del tipo *EMF*.

***Acceleo*[[11]](#footnote-11)**

*Acceleo* es una herramienta de transformación *M2T* basada en los estándares propuestos por la *OMG* y que actualmente forma parte de la *Eclipse Foundation*.  *Acceleo* es el resultado de varios años de investigación y desarrollo en el área de los lenguajes de transformación de modelos). Permite la des-serialización de modelos basados en *UML* del *EMF* como así también modelos basados en el metamodelo *Ecore*. *Acceleo* posee una herramienta de desarrollo bastante madura como así también una comunidad activa que la sostiene. Muchos proyectos en la industria han probado su eficacia en varios contextos.

**3.3 Beneficios de los lenguajes de transformación**

Los lenguajes de transformación *M2T* separan el código estático y dinámico utilizando el enfoque de plantillas (*plantillas)* para implementar las transformaciones *M2T*. Una plantilla puede ser vista como un proyecto que define elementos de texto estáticos compartidos por todos los artefactos, como así también, partes dinámicas que deben ser completadas con información específica para cada caso en particular. Por lo tanto, un *template* contiene fragmentos de texto simples para las partes estáticas y los llamados metamarcadores (*meta-markers)* para las partes dinámicas. Los metamarcadores son marcadores de posición y deben ser interpretados por un motor que procesa los *plantillas* y consulta fuentes de datos adicionales para producir las partes dinámicas. Las fuentes adicionales de datos son los modelos. En la se presenta el esquema tradicional de transformación basado en *plantillas.*

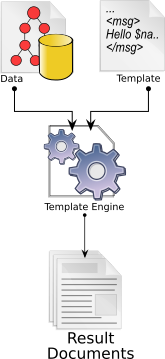


Figura 3 Plantilla, motor de plantillas y modelos de entrada para producir texto

Los *plantillas* permiten representar explícitamente la estructura del texto de salida dentro del *template*. Esto permite una especificación de la generación de código más entendible y leíble.

En los metamarcadores, el código es utilizado para acceder a la información almacenada en los modelos.  El estándar *OCL* es la elección para llevar a cabo esta tarea en la mayoría de los lenguajes de transformación basados en *plantillas*. De esta forma, los lenguajes de transformación *M2T* también permiten el uso de *OCL* (o dialecto de *OCL*) para especificar a los metamarcadores.

Los lenguajes de transformación *M2T* actuales vienen con soporte de herramienta, lo cual permite leer directamente los modelos y serializar texto en archivos, definiendo solamente archivos de configuración.

**3.2 Principales enfoques de desarrollo *Web* basado en modelos para las *RIA***

En [] y [] se identifica la necesidad de metodologías sistemáticas para el desarrollo de las *RIA* y se llevan a cabo estudios presentando las diversas metodologías *Web* existentes para ese fin. Un estudio más exhaustivo y reciente de comparativas se presenta en [] en donde se clasifican las metodologías en las siguientes categorías:

1. Contribución a la investigación proveniente de la comunidad de ingeniería *Web*, derivada de la evolución de los enfoques dirigidos por modelos concebidos para el diseño y desarrollo de aplicaciones *Web* tradicionales. Esta categoría incluye a *WebMl*-*RIA*[<fraternali2010>], *OOHDM*-*RIA*[], *OOH4RIA*[], (*UWE-R*[], Patrones con *UWE*[] y *UWE*+*RUX* []).
2. Enfoques de desarrollo sistemáticos provenientes de la comunidad de *Human Computer Interaction (HCI),* en donde el diseño *RIA* es el foco principal. Las metodología *RUX* [] y *UsiXML*[<martinez-ruiz2010>] []. pertenecen a esta categoría
3. Enfoques que combinan *HCI* y técnicas de ingeniería *Web*: espacios interactivos con *UML* presentado en [] y *OOWS for* *RIA* [].
4. Propuestas recientes de los vendedores de herramientas comerciales que adoptan *MDD*, entre ellos *WebRatio*, *Mendix*, *Novulo*, RUX-Tool y *Thinkwise*.

Con respecto al contexto en el cual este trabajo analiza las metodologías *Web* anteriores, una de las consideraciones que se ha tenido en cuenta es que las mismas adopten estándares (por ejemplo, *UML*). También se ha buscado que las metodologías en cuestión sean de uso abierto para la comunidad de desarrolladores y no propietarias. Un análisis más profundo de las metodologías de la categoría d) del estudio mencionado no se ha considerado en este trabajo debido a que son propuestas cerradas basadas en herramientas comerciales. He ahí que a continuación se describirán brevemente las metodologías basadas en *UML*: *OOH4RIA*, *UWE-R*, Patrones con *UWE*, *UWE* combinado con la herramienta *RUX* y los patrones de interacción con *OOWS*. Las demás metodologías se presentarán en un cuadro comparativo con sus respectivos alcances para las *RIA*.

**3.2.1 Extensión a *OO-H* (*OOH4RIA*)**

OO-H (*Object Oriented Hypermedia*) [] es una metodología orientada a objetos para la *Web* tradicional, basada parcialmente en estándares (*XML*, *UML* y *OCL*). Se propone a esta metodología en su forma original, un enfoque *MDD* para especificar una aplicación *RIA*, por medio de una extensión, agregando nuevos modelos para la presentación. La muestra una representación del proceso *MDD* con las definiciones de modelos y transformaciones que permiten obtener la implementación correspondiente a las *RIA*, como así también, los actores que participan en el ciclo de desarrollo. *OOH4RIA*, propone un metamodelo de presentación definido con abstracciones de bajo nivel, donde los elementos principales son representados por los *widgets* proveídos por una plataforma específica, en este caso *Google Web Toolkit* (*GWT*)[[12]](#footnote-12). Este metamodelo permite la especificación de los aspectos estructurales de las *RIA*. Los *widgets* pueden ser combinados, extendidos, adaptados y enlazados a otros modelos. Se genera el código de la aplicación tanto para el lado cliente como para el lado servidor.

Figura 4 Representación del proceso *MDD* para *OOH*-RIA. Fuente: S Meliá y otros, 2008

**3.2.2 Extensiones *RIA* a *UWE* (*UWE-R*)**

El enfoque *UWE* (*UML-based Web Engineering)* [][<koch2001>] es una metodología orientada a objetos que tiene la particularidad distintiva de que está basada totalmente en *UML*. Está definida en la forma de perfil y de por sí, es una extensión al metamodelo *UML*. *UWE-R* es una ligera extensión de *UWE* para *RIA*, que abarca las capas de navegación, proceso y presentación. Por lo tanto, los nuevos elementos de modelado están definidos heredando la estructura definida y el comportamiento de los elementos *UWE*.

Con respecto a las extensiones a la navegación, se extienden las metaclases *Nodo* y *Enlace*. Como puede verse en la , la metaclase *Nodo* es extendida agregando la metaclase *RichNavigationClass*, que a diferencia de *UWE* clásico, que se basa en la navegación hipertextual principalmente, esta nueva metaclase podría estar contenida dentro de un objeto *Flash* o un *Java Applet*. La metaclase *Enlace* se extiende agregando la metaclase *RichNavigationLInk*, que tiene como finalidad modelar la interacción entre la aplicación cliente y servidor, especificando si se trata de una comunicación síncrona o asíncrona. En el caso de ser asíncrona, la respuesta es un *callback*.



Figura 5 Extensiones a las metaclases Nodo y Enlace. Fuente: Machado y otros, 2009

A nivel de presentación, se agregan algunas metaclases para expresar la riqueza de las *RIA* con respecto al aspecto de la interfaz de usuario, como puede verse en la .



Figura 6 Extensiones al metamodelo de Presentación en *UWE-R*. Fuente: Machado y otros, 2009

Por último, se llevan a cabo extensiones con respecto al proceso (o la lógica de la aplicación), con lo cual, se pueden modelar los procesos que pueden realizarse en los lados servidor y cliente respectivamente. También por medio de la metaclase *Autonomous Action* se pueden modelar las acciones que deben llevarse a cabo automáticamente en la aplicación, sin la interacción del usuario, por ejemplo, que en el caso de que expire un temporizador, se dispare automáticamente alguna acción.

**3.2.3 *UWE* combinada con la herramienta *RUX***

La metodología *UWE* también puede combinarse con el método *RUX* []. La metodología *RUX* es un enfoque dirigido por modelos para el enriquecimiento de las interfaces de usuario. Puede ser utilizada en el tope de muchas metodologías de modelado *Web*. En este enfoque, *UWE* es utilizado para especificar el contenido, navegación y proceso de negocio de una aplicación *Web*, y la metodología *RUX* se emplea sobre estos modelos para adicionar capacidades enriquecidas a la interfaz de usuario. En esta propuesta se busca construir el puente entre ambos enfoques, definiendo reglas de transformación entre sus respectivos metamodelos. En otros términos, se extienden las reglas de generación de *UWE* de manera a obtener la conexión con la metodología *RUX* automáticamente.

La metodología *RUX* presenta 3 niveles de interfaces, proveyendo de esta forma una cadena de refinamientos. La interfaz abstracta provee de una representación común a todos los dispositivos y plataformas de desarrollo *RIA*, sin ningún tipo de dependencia espacial, de estética ni de comportamiento. La interfaz concreta es independiente de la plataforma, pero específica para un dispositivo o grupo de dispositivos. Está dividida en 3 niveles de presentación, espacial, temporal y presentación interactiva. En la presentación espacial, los modeladores simplemente necesitan refinar esta agrupación, especificar el arreglo espacial de los componentes y definir sus dimensiones y la estética. La presentación temporal permite la especificación del comportamiento, lo cual requiere una sincronización temporal (por ejemplo, animaciones). La presentación interactiva permite la especificación del comportamiento del usuario con la interfaz de usuario *RIA*.

La interfaz final contiene la información final para la generación de código de la interfaz de usuario, lo cual es específico para un dispositivo o un grupo de dispositivos y para una plataforma de desarrollo *RIA* tal como *Flex*, *Ajax* o *OpenLaszlo*[[13]](#footnote-13).

**3.2.4 *UWE* con patrones**

*UWE* puede extenderse por medio de patrones[].Los patrones *RIA* describen la interacción, la operación y la presentación de un *widget*, en donde la interacción es el disparador del patrón *RIA* (por ejemplo, el movimiento del mouse, presionar una tecla o un evento temporal). Como resultado de la interacción, una variedad de operaciones pueden ser llevadas a cabo, tales como validaciones, búsquedas y refrescados de página. Finalmente el resultado implica una actualización en la interfaz de usuario.

Este enfoque consiste en el uso de modelos que representan *widgets* *RIA*, y la inclusión de estos modelos dentro de metodologías de desarrollo *Web* existentes. Cada *widget* es modelado por medio de máquinas de estados que representan la característica *RIA* deseada. En la Figura 7 se muestran los patrones definidos en el trabajo de Koch[<koch2009>].

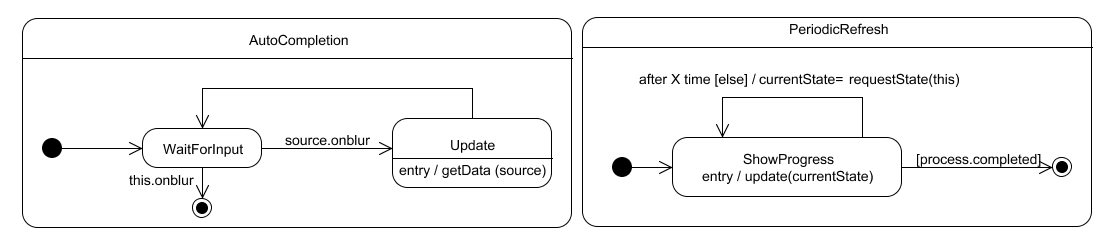
****

Figura 7 Patrones *UWE*. A la izquierda el patrón *Autocompletion*. A la derecha el patrón *Periodic Refresh*. Fuente Koch y otros, 2009

**3.2.5 Patrones en *OOWS***

La principal contribución de patrones con *OOWS*, es un modelo de interacción para especificar la nueva semántica para hacer frente al desarrollo basado en modelos *RIA* [<valverde2008>]. El modelo se compone de patrones de interacción que describen, desde el punto de vista conceptual, una solución genérica para la interacción común de un usuario con un sistema siguiendo los principios de la *Human Computer Interaction (HCI)[[14]](#footnote-14)*. Este modelo se basa en los siguientes aspectos: 1) una vista abstracta, que consta de patrones de interacción abstractos, que describen la interacción sin tener en cuenta los detalles tecnológicos, y 2) una vista concreta, formada por patrones de interacción *RIA* que especifican la nueva interacción y los requerimientos para la interfaz. Con estas dos premisas se implementan *widgets* para el autocompletado y la expansión/colapso de texto. En la Figura 8 se puede analizar el proceso de desarrollo para esta propuesta.



Figura 8 Un resumen del enfoque *MDD* con patrones para *OOWS*. Fuente: Valverde y otros, 2008

La Tabla 1 que se muestra a continuación presenta un resumen de las propuestas, indicando las características *RIA* contempladas por las mismas que fueron expuestas en el Capítulo 2 del libro.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Características versus metodologías | | *OOHDM*-*RIA* | *OOH4RIA* | *WebMl* - *RIA* | Patrones con *UWE* | Patrones *OOWS* | *UsiXML* | *UWE-R* | Espacios interactivos con *UML* | *UWE* + *RUX* |
| Almacenamiento en el lado del cliente | | - |  | si | - | - | - | - | - | - |
| Lógica de negocio en el lado del cliente | Operaciones complejas | - | - | si | - | - | - | - | - | - |
| Operaciones específicas del dominio | - | - | - | - | - | - | si | - | - |
| Validación local | si | si | si | - | - | - | - | - | si |
| Presentaciones enriquecidas | Manejo de eventos en el lado cliente | - | - | si | si | si | - | si | si | si |
| Widgets | si | si | - | si | si | si | si | si | si |
| Paradigma de página única | si | si | si | si | - | si | - | - | si |
| Contenido multimedia | - | si | - | - | - | si | si | - | si |
| Comunicación cliente servidor | Sincronización de datos | - | - | si | - | - | - | si | - | si |
| Obtención de actualizaciones parciales de página | si | si | si | si | si | - | si | si | si |
| Push y Pull | - | - | si | - | - | - | si | - | - |

Tabla 1 Metodologías *Web* y sus alcances para *RIA*

En el análisis de la Tabla 1, se nota que la metodología que más características de las *RIA* abarca es *WebMl*, con la salvedad que utiliza herramientas propietarias para su modelado[[15]](#footnote-15), se basa en un *DSL* gráfico propio, no utiliza *UML* y no cubre widgets. Con respecto a la característica de presentaciones enriquecidas, que es la que concierne a este trabajo de tesis, la metodología *RUX* y la combinación de *UWE*+*RUX* son las que ofrecen cobertura completa a diferencia de las otras metodologías. Sin embargo, *RUX* no es precisamente una metodología, sino más bien una herramienta propietaria que sirve para enriquecer con características de las *RIA* a las metodologías *Web*. *UsiXML* ofrece una metodología estándar bastante completa que utiliza una serie iterativa de transformaciones XSLT *(Extensible Stylesheet Language Transformations)* para obtener la interfaz de usuario final para una plataforma destino a partir de una interfaz abstracta, definida previamente, pero está abocada específicamente al desarrollo de interfaces y no es una metodología que abarque todo el ciclo de vida de una aplicación *Web*.

Dado el comportamiento dinámico y reactivo de los *widgets* es necesario representarlos con diagramas que logren captar su dinamismo. He allí que las metodologías más influyentes en este trabajo son *UWE-R*, *UWE* con patrones, los espacios interactivos con *UML*, *OOHDM*, *OOWS* y *OOH*-4RIA que proponen diagramas interactivos (de estado y de secuencia) para la representación de los elementos interactivos, necesarios en las presentaciones de *Web* 2.0.

**3.3 La Aproximación *MoWebA* (Model Oriented *Web* Approach)**

*MoWebA* [][<gonzalez2011>] es una propuesta creada en el DEI (Departamento de Electrónica e Informática) que adopta los principios de *MDA*. En la Figura 9 se muestran las dimensiones de *MoWebA*. Como puede observarse, consta de fases, niveles y aspectos, que se van describiendo a continuación.

Las fases se refieren a los procesos de modelado y transformación. Estas se encuentran claramente diferenciadas e incluyen a su vez una serie de modelos. Las fases y modelos se describen a continuación:

1.  **Modelado del problema:** incluye al *CIM* (*Computation Independent Model*), orientado al modelado de los requisitos funcionales*,* y al *PIM* (*Platform Independent Model*), orientado al modelado del problema sin considerar aspectos de la arquitectura o plataforma. A partir de aquí es posible llevar a cabo transformaciones para obtener los modelos específicos de la plataforma de manera semi-automática por medio de reglas de transformación.

2.   **Modelado de la solución**: incluye al *ASM* (*Architectural Specific Model*) y al *PSM* (*Platform Specific Mode*l). Es en esta fase en donde todos los detalles de la arquitectura y plataforma destino se definen, permitiendo generar a partir de aquí, el código de la aplicación de manera automática. En *MoWebA* se independiza esta fase, y esto hace que sea bastante prometedora para la implementación de las *RIA*, debido a que existen numerosas plataformas destino para desplegarlas. En las aproximaciones estudiadas, por lo general las extensiones *RIA* son definidas en el marco de los modelos conceptuales (*PIM*), haciendo que los modelos que deberían ser independientes de la solución, adquieran elementos que ya son propios de una arquitectura específica.



Figura 9 Fases, niveles y aspectos en el desarrollo de MoWebA.

3. **Código fuente:** incluye al *ISM* (*Implementation specific model*), que corresponde al código generado y el código manual a ser agregado (en caso de ser necesario) para generar la aplicación final. La aplicación puede refinarse, dado que todas las fases son iterativas e incrementales.

*MoWebA* también presenta distintos niveles de construcción separados en capas para representar a una aplicación *Web*. Se contemplan niveles para el contenido, la lógica del negocio, la navegación, la presentación y los usuarios. Los aspectos están relacionados con la estructura y el comportamiento de la aplicación. Cada modelo es visto desde dos puntos de vista (estructura y comportamiento), por lo que existe una propuesta notacional para definirlos.

Definir una propuesta *RIA* para *MoWebA* resulta interesante ya que sería posible realizar un análisis para diferenciar el *PIM* del *ASM*, aspecto no contemplado en otras metodologías. Esto hace que al definir los modelos propios de las *RIA*, si hubiera necesidad de llevar a cabo una migración a otra arquitectura destino, probablemente deberán realizar muchos cambios sobre el modelo mismo. En *MoWebA* se plantea tener siempre el mismo *PIM*, y a partir de este adoptar la arquitectura correspondiente.

**3.3.1 La capa de presentación de *MoWebA***

La capa de presentación de *MoWebA* abarca a los metamodelos de contenido y estructura (ver Figura 10). En el metamodelo de contenido (*Content*) se tienen los diversos elementos de interfaz (*uIElements*) correspondientes a la *Web* 1.0. Entre estos elementos están los *textInput*; los enlaces, que podrían corresponder a navegaciones internas de la aplicación (*anchor*), o bien a navegaciones externas (*externalLink*); los *button;* los elementos del tipo selección que corresponden a los *choice*  y a los *dropBox*; los *text,* para texto plano en las páginas; *htmlText,* para el despliegue de cualquier texto *HTML*; y el elemento del tipo *multimedia* para audio y video. Cada uno de estos elementos cuenta con sus respectivos *atributos* para identificar a sus propiedades intrínsecas. El *compositeUIElement* contiene a los distintos *uIElements* y en él pueden definirse condiciones de tipo *orderBy* y *groupBy,* en caso que sea necesario obtener datos del modelo de dominio. El elemento de interfaz *form* extiende al *compositeUIElement*, permitiendo definir a los distintos uIElementsdentro de un formulario de entrada. El elemento de interfaz *table* contiene a los atributos *rows y columns* para establecer la cantidad de filas y columnas que contendrá la tabla, para desplegar a los distintos elementos de interfaz que pueden ser *uIElement* o *compositeUIElement*.

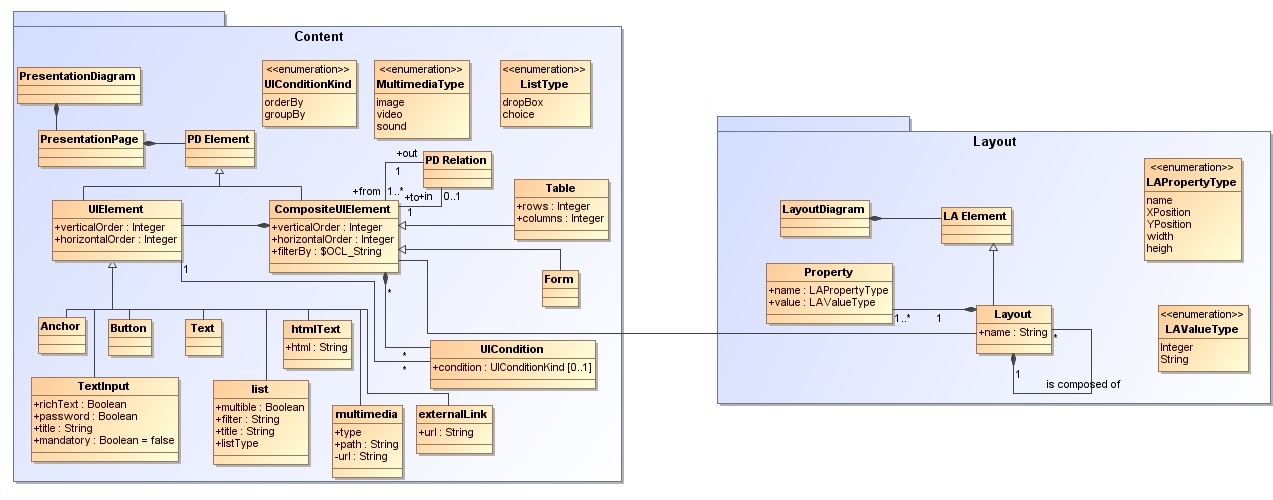


Figura 10 Metamodelo de Contenido y Estructura de *MoWebA*

El metomodelo de estructura (*Layout*) permite establecer a cada uno de los *compositeUIElement* definidos en el metamodelo de contenido,una posición específica dentro de las páginas. Un *Layout* está compuesto de uno o muchos *Layout*, y cada uno de ellos a la vez puede tener una o varias propiedades definidas, que corresponden a sus coordenadas posicionales.

Los metamodelos *Content* y *Layout* definen la sintaxis abstracta de la capa de presentación de *MoWebA* por medio del estándar *MOF* (*Meta Object Facility*). La sintaxis concreta de *MoWebA* es llevada a cabo por medio de *UML*, utilizando la técnica de perfil (*profiling*), que permite agregar a *UML* los diversos estereotipos *(stereotypes)* y valores etiquetados (*tagged values*) propios de *MoWebA*. Los perfiles *Content y Layout* permiten definir los *PIM* de presentación de una aplicación modelada con *MoWebA*. Los perfiles de Contenido y Estructura se presentan en la Figura 11 y en la Figura 12 .

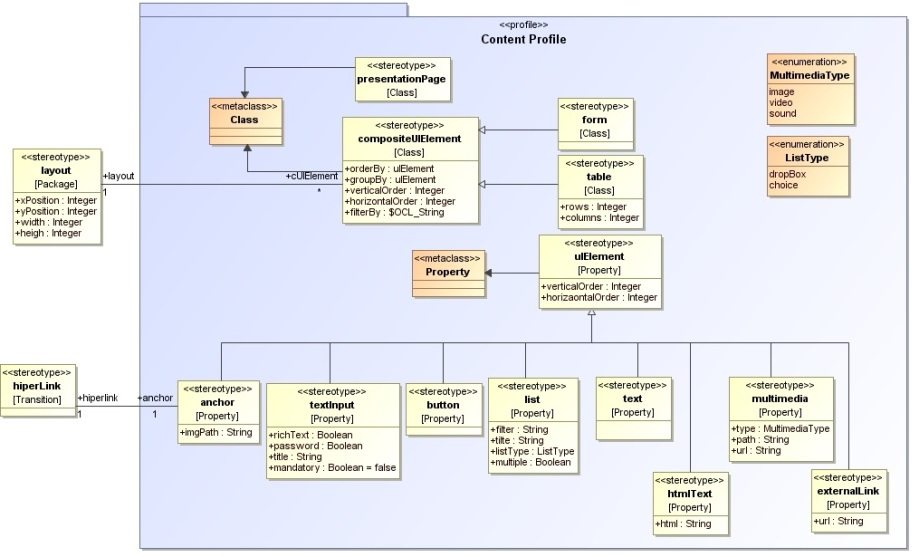


Figura 11 Perfil de Contenido de *MoWebA*

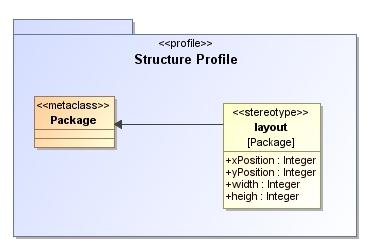


Figura 12 Perfil de estructura de *MoWebA*

A modo de ejemplo, se presenta en la Figura 13, el *PIM* correspondiente a la presentación de una aplicación con *MoWebA*, en la que se solicita el ingreso de datos personales, utilizando para el modelado, el perfil de Contenido y el de Estructura. En la Figura 14, se presenta la interfaz de usuario obtenida a partir del *PIM* de la Figura 13.

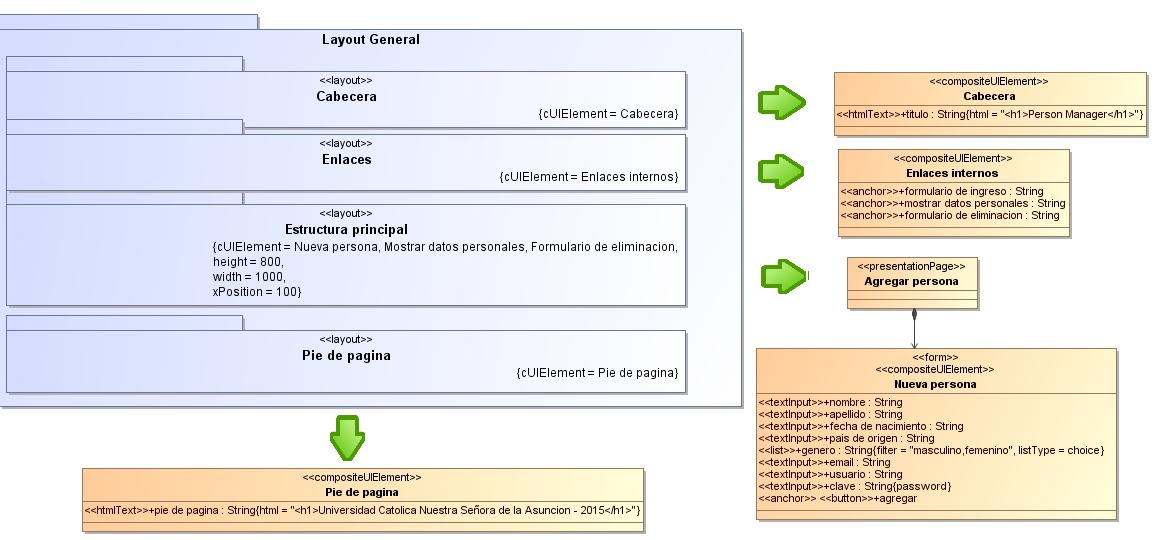


Figura 13 *PIM* modelado con el perfil de contenido de *MoWebA*

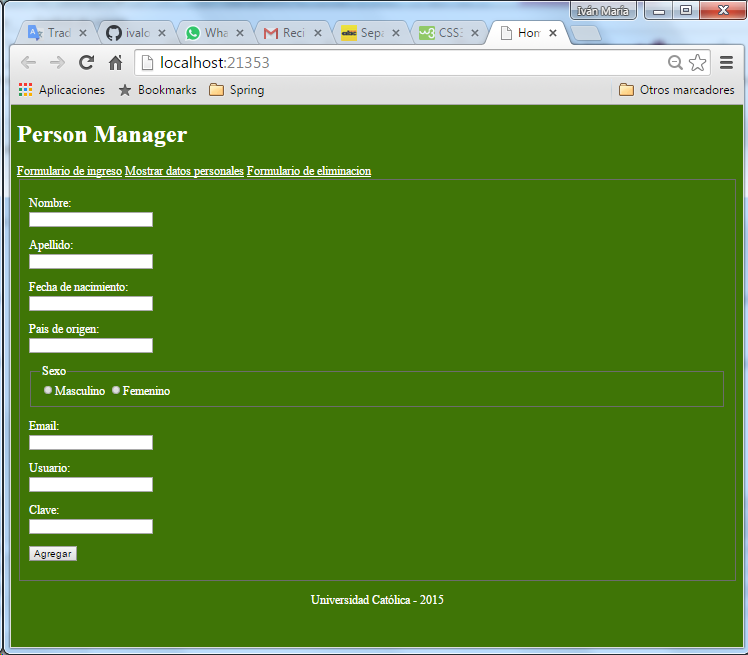


Figura 14 Interfaz obtenida a partir del *PIM* presentado en la Figura 14

**3.3.2 El enfoque utilizado con *MoWebA* para la generación de interfaces**

La Figura 15 representa el proceso para el modelado y generación de interfaces con *MoWebA*. Primeramente, se modelan los *PIM* que representan a una aplicación en particular utilizando distintos perfiles *UML* de *MoWebA*. Estos perfiles representan extensiones a *UML* para agregar características específicas de *MoWebA* a los metamodelos, para que de esta forma sea posible representar la sintaxis concreta del *DSL* (*Domain Specific Lenguage*). Los modelos *PIM* y los perfiles están basados en el estándar *MOF* que forma parte del enfoque *MDA*. Los modelos *PIM* se crean utilizando la herramienta *MagicDraw*[[16]](#footnote-16). Posteriormente, tanto los *PIM* como los perfiles son exportados al formato *XMI* del *EMF*[[17]](#footnote-17). Esto de por sí es llevado a cabo a fines de tener compatibilidad con la herramienta de transformación *M2T* *Acceleo*[[18]](#footnote-18), que toma como entrada modelos *UML* que están basados en el metamodelo *Ecore*[[19]](#footnote-19).

Una vez exportados los modelos (*PIM* y profile) al *Acceleo*, por medio de las plantillas de transformación y los módulos de servicio en *Java* (*Java Service Wrappers*), que forman parte de *Acceleo*, es posible llevar a cabo las transformaciones necesarias sobre los modelos de entrada para obtener los archivos fuentes (.*HTML* y .*CSS*) que representan a la aplicación en sí. Las plantillas de transformación permiten establecer la estructura del código fuente que va a generarse, estableciendo las partes estáticas (código que va a generarse en ciertas condiciones y que no cambia) y dinámicas (código que es obtenido a partir de los modelos de entrada). Por medio de los metamarcadores de las plantillas de transformación de *Acceleo*, es posible definir expresiones *OCL* para la manipulación de los distintos elementos definidos en el modelo de entrada. Los módulos de servicio *Java* permiten complementar a las plantillas de transformación, dando la posibilidad de agregar código *Java* para la manipulación de los elementos pertenecientes a los modelos.

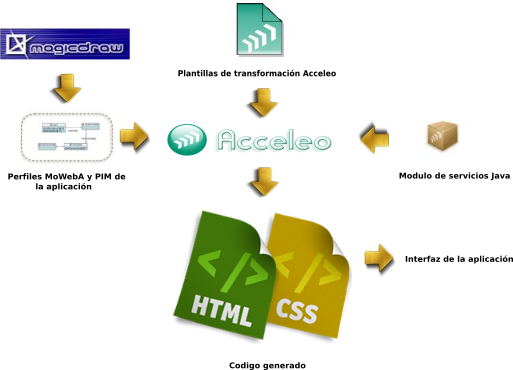


Figura 15 Fases para la generación de interfaces con *MoWebA*

**3.7 Resumen del Capítulo**

Los modelos hoy en día ocupan un lugar importante en el proceso de desarrollo de software, comúnmente para la comunicación entre los desarrolladores y las personas sin conocimientos técnicos, o bien entre los mismos desarrolladores. Las metodologías de desarrollo enmarcadas en el contexto *MDD* y *MDA* toman estos modelos y por medio de transformaciones sobre los mismos (aplicando técnicas *M2M* o *M2T)*, pueden obtener el código fuente de una aplicación para una plataforma destino en particular. Son varias las propuestas metodológicas *Web* existentes en la actualidad en donde los modelos y las transformaciones sobre estos son los elementos principales del proceso. Algunos ejemplos son: *OOHDM*-*RIA*, *OOH4RIA*, *WebMl*-*RIA*, Patrones con *UWE*, Patrones *OOWS*, *UsiXML*, *UWE-R*, Espacios interactivos con *UML* y *UWE* + *RUX*. Se ha visto que muchas de ellas cubren características de las *RIA* en ciertos aspectos, pero ninguna ofrece cobertura total a todas las características. Es por ese motivo, que resulta necesario extender alguna de las metodologías *Web* existentes o bien crear nuevas metodologías *Web* para satisfacer esta necesidad.

La metodología *Web* *MoWebA*, resulta prometedora para llevar a cabo extensiones y de esta forma dar cobertura a características de las *RIA*, debido principalmente a la forma en la que está estructurada la metodología, en donde existe una separación adecuada de conceptos y capas para el modelado de una aplicación, a la par de contemplar todo el ciclo de desarrollo de una aplicación.

1. **MDA** [www.omg.org/mda/](http://www.omg.org/mda/) 2015 [↑](#footnote-ref-1)
2. ***UML*:** [www.omg.org/spec/*UML*](http://www.omg.org/spec/UML) 2015 [↑](#footnote-ref-2)
3. **OCL:** [www.omg.org/spec/OCL](http://www.omg.org/spec/OCL) 2015 [↑](#footnote-ref-3)
4. ***MOF*:** [www.omg.org/*MOF*/](http://www.omg.org/mof/) 2015 [↑](#footnote-ref-4)
5. **QVT:** [www.omg.org/spec/QVT/1.1/](http://www.omg.org/spec/QVT/1.1/) 2015 [↑](#footnote-ref-5)
6. **XSLT:** [www.w3.org/TR/xslt20/](http://www.w3.org/TR/xslt20/) 2015 [↑](#footnote-ref-6)
7. **XMI:** <http://www.omg.org/spec/XMI/> 2015 [↑](#footnote-ref-7)
8. **JET:** <https://projects.eclipse.org/projects/modeling.m2t.jet> 2015 [↑](#footnote-ref-8)
9. **XPAND:** <https://eclipse.org/modeling/m2t/?project=xpand> 2015 [↑](#footnote-ref-9)
10. **MOFScript:** <https://eclipse.org/gmt/mofscript/> 2015 [↑](#footnote-ref-10)
11. ***Acceleo*:** [www.*Acceleo*.org/](http://www.acceleo.org/) 2015 [↑](#footnote-ref-11)
12. **Google *Web* Toolkit:** <http://www.gwtproject.org/> 2015 [↑](#footnote-ref-12)
13. **OpenLaszlo:** <http://www.openlaszlo.org/> 2015 [↑](#footnote-ref-13)
14. **Interaction Design Foundation:** <http://www.interaction-design.org/encyclopedia/human_computer_interaction_hci.html> 2015 [↑](#footnote-ref-14)
15. **WebRatio:** <http://www.webratio.com/site/content/es/home> 2015 [↑](#footnote-ref-15)
16. **No Magic:** [http://www.nomagic.com/products/*MagicDraw*.html](http://www.nomagic.com/products/magicdraw.html) 2015 [↑](#footnote-ref-16)
17. **Eclipse Modelling Framwwork:** <https://www.eclipse.org/modeling/emf> 2015 [↑](#footnote-ref-17)
18. ***Acceleo*:** [https://eclipse.org/*Acceleo*](https://eclipse.org/acceleo) 2015 [↑](#footnote-ref-18)
19. **Ecore:** Metamodelo nativo que forma parte del core del EMF para describir a los modelos [↑](#footnote-ref-19)