Universidad Católica

“Nuestra Señora de la Asunción”

Facultad de Ciencias y Tecnología

***Departamento de Ingeniería Electrónica e Informática***



**Proyecto Final**

*“****Una propuesta MDA para el soporte de aplicaciones RIA****”*

**Alumno: Iván López.**

**Coordinadores: Ing. Magalí González.**

**Msc. Ing. Nathalie Aquino.**

**Asunción**

**2015**

**Agradecimientos**

*A mi familia por su amor incondicional.*

*A mis tutoras de tesis, por su paciencia y ayuda desinteresada en todos estos años de trabajo.*

*A mi maestro de vida Daisaku Ikeda, por sus orientaciones e incansable aliento.*

Indice

Indice de figuras

**Capitulo 1**

**introducción**

El número de usuarios con acceso a Internet, ha crecido de manera exponencial y sigue en aumento[[1]](#footnote-1). Internet se ha vuelto un servicio básico como el agua, la luz o el gas. Esto se debe por un lado al incremento en el ancho de banda de las conexiones de red y el abaratamiento en los costes de los servicios. Por otro lado, el mundo globalizado actual, exige en cierto sentido, la necesidad de acceder de manera ágil a la información, a la vez de acortar la distancia física entre los usuarios; permitiendo de esta forma mantener una comunicación en tiempo real con personas ubicadas en puntos muy distantes del mundo, compartiendo recursos varios, tales como datos multimedia, archivos, etc. De igual forma la red de redes Internet, se ha convertido en una herramienta fundamental para la búsqueda de información de todo tipo[[2]](#footnote-2), para fines educativos[[3]](#footnote-3), sociales[[4]](#footnote-4),[[5]](#footnote-5) y el entretenimiento[[6]](#footnote-6). Lo que un tiempo atrás se limitaba exclusivamente a fines académicos, hoy en día está al alcance de toda la sociedad.

He allí que las aplicaciones web toman un rol protagónico, debido a que los usuarios demandan mejores aplicaciones, que sean más interactivas y que ofrezcan funcionalidades naturalmente intuitivas y ágiles. De alguna forma, esta demanda se ha podido lograr, gracias a la ingeniería web que define el uso de procesos científicos y principios de administración, acompañado de enfoques sistemáticos, con la meta de desarrollar, desplegar y mantener satisfactoriamente una alta calidad en los sistemas y aplicaciones basados en Web [<ginige2001>]. Muchas empresas de software, han tenido éxito desarrollando aplicaciones web complejas y de alta performance. Pero así también, otras no han tenido éxito. Las causas fundamentales del porqué de estos fallos se deben principalmente a una falta de visión, metas ciegas, procesos de diseño y desarrollo defectuosos, acompañados de una pobre administración en el esfuerzo de desarrollo, siendo no la tecnología propiamente dicha la raíz de las fallas. Es por eso que la esencia de la ingeniería web se basa en administrar adecuadamente la diversidad y complejidad en el desarrollo de las aplicaciones web evitando así, fallas potenciales que pueden llevar a tener serias implicancias.

Con la idea de que las aplicaciones web se asemejen lo más posible a las aplicaciones de escritorio, nacieron las *Rich Internet Applicantions* (RIAs). Estas representan todo un desafío para la ingeniería web, ya que las RIAs han dado un cambio radical en la manera en que se comportan, desarrollan y despliegan las aplicaciones web, ofreciendo mejoras substanciales con respecto a las aplicaciones web tradicionales, con nuevas características referentes a la comunicación, la distribución de los datos y la computación en el lado cliente, acompañadas de interfaces mucho mas interactivas, en donde el usuario en ocasiones, no distingue si está utilizando la aplicación online o offline. Con estos avances propuestos por RIAs, muchas de las metodologías web tradicionales basadas en la web 1.0, tales como; WebMl[<fraternali2010>], UWE[<machado2009>], OOH[<melia2008>], OOHDM[<urbieta2007>] y OOWS por citar algunas, han tenido que evolucionar de cierta forma, agregando nuevos modelos o extendiendo los existentes, para dar cobertura a las diversas características sofisticadas propuestas por RIAs. Muchas de las metodologías citadas han logrado una notable evolución en su afán de mantenerse vigentes con los avances propuestos por las RIAs, sin embargo en la actualidad, ninguna de ellas, logra satisfacer todas las nuevas funcionalidades [<wright2008>] [<preciado2005>] [<mariannebusch2009>] [<toffetti2011>].

He allí que resulta necesario crear nuevas metodologías de desarrollo web u bien extender las actuales para satisfacer las nuevas características impuestas por la tendencia actual. Con ese propósito, una nueva metodología para el desarrollo de aplicaciones web basada en modelos y fundamentada en los principios propuestos por la OMG[[7]](#footnote-7), se ha propuesto en el DEI[[8]](#footnote-8). Esta metodología está basada en los estándares MDA[[9]](#footnote-9) y ofrece un esquema de modelado en capas para la separación de conceptos. La metodología se denomina MoWebA[<gonzalez2011>] [<gonzalez2010>], y en la actualidad cuenta con características de modelado a nivel de presentación, lógica de negocio, navegación y adaptabilidad de los usuarios, pudiendo generarse aplicaciones web completas y funcionales con modelos independientes de la plataforma. MoWebA tiene la capacidad de llevar a cabo extensiones a sus metamodelos para cubrir nuevas características, lo cual la hace adaptable a los cambios actuales.

El objetivo de este trabajo de fin de carrera se enmarca en la idea de llevar a cabo extensiones a la metodología web MoWebA con respecto a la capa de Presentación, con el fin de abarcar a algunas de las principales características de las *Rich Internet Applications*. Para llevar a cabo esta propuesta de extensión, se dividirá el esquema de trabajo de la siguiente manera.

* Se definirán las *Rich Internet Applications* (RIAs), presentando sus principales características y los nuevos aportes a las aplicaciones Web tradicionales.
* Se dará a conocer las principales metodologías de desarrollo basada en modelos (MDD) que dan cobertura a características de RIAs.
* Se presentará la aproximacións de desarrollo web MoWebA.
* Se extenderá al metamodelo de contenido de MoWebA y se propondrá una propuesta de transformación de modelo a texto (M2T) para la plataforma destino *jQueryUI* y *JQuery Validation Plugin*, para cubrir algunas características RIAs de las presentaciones enriquecidas y de la lógica de negocios en el lado cliente.
* Se validará el metamodelo propuesto con un caso de estudio.
* Se finalizará el trabajo con un análisis de los resultados obtenidos elaborando la conclusión y los posibles trabajos futuros.

**Capitulo 2**

**Marco teórico de las rich internet applications**

**2.1 Las Rich Internet Applications (RIAs)**

Desde el lanzamiento oficial del primer sitio web en 1991 por Tim Berners Lee hasta hoy en día, las aplicaciones web que forman parte de la red de redes internet, han evolucionado de la web 1.0, en la que los usuarios obtenían información estática representada en documentos hipertextuales, a la web 2.0, en la cual la información de las páginas es generada de manera dinámica y en la que se combinan, no solamente información textual, sino también, características multimedia en las interfaces (audio, video streaming, widgets interactivos, entre otros). De igual forma, la evolución en la web también vino acompañada de cambios tecnológicos en los diferentes navegadores web y en los distintos protocolos de comunicación entre las aplicaciones cliente y servidor.

Muchos de estos avances se dieron, debido a las limitaciones de las aplicaciones web tradicionales en cuanto a la usabilidad e interactividad que ofrecen sus interfaces de usuario. Esto se debe en gran medida, a la comunicación síncrona existente entre el cliente y el servidor en la que, por cada acceso a un enlace, el cliente debe esperar la respuesta del servidor que una vez obtenida, debe recargar la página completamente, quedando ocioso la mayor parte del tiempo []. He ahí que surgen, como alternativa, las denominadas Aplicaciones de Internet Enriquecidas (*Rich Internet Applications* - RIAs) con la idea de mejorar las aplicaciones web tradicionales, agregando nuevas características que se encuentran presentes en las aplicaciones de escritorio. El término fue introducido en marzo de 2002 por la empresa Macromedia (actualmente Adobe) que en ese entonces abordaba las limitaciones en cuanto a la riqueza de las interfaces, medios y contenidos de las aplicaciones [].

Dado que las RIAs poseen numerosas características innovadoras, es difícil ofrecer una definición formal que englobe todos sus atributos. Diversos autores la han enmarcado en un contexto particular y todas las citadas aquí resultan valederas. A continuación se presentan algunas descripciones interesantes de algunos autores:

“Las RIAs, emulan características de las aplicaciones de escritorio, mejorando la experiencia de los usuarios con nuevos efectos visuales, dándose principal realce a las características multimedia. El intercambio de los datos puede llevarse a cabo por medio de una comunicación asíncrona, de tal forma que el cliente permanece receptivo a eventos, mientras que continuamente recalcula o actualiza partes de la interfaz de usuario. Las RIAs se caracterizan por poseer una variedad de controles interactivos de operación (widgets), y por dar la posibilidad de utilizar la aplicación con o sin conexión al servidor (uso offline de la aplicación), y también por ofrecer un uso transparente de las capacidades del cliente, del servidor y de la conexión de red” [].

“En las RIAs, las aplicaciones se cargan de manera completa en el cliente, desde el inicio, realizándose la comunicación con el servidor solamente en caso de que sea necesario actualizar los datos desde una base de datos o bien desde un archivo externo. La navegabilidad de las aplicaciones web mejora de manera substancial, debido a que se evitan las recargas innecesarias de toda la página, actualizando solamente las porciones de ésta que son relevantes. Con esto se minimiza la cantidad de información que se transmite por la red a la par de mejorar la performance de la aplicación[[10]](#footnote-10)”.

“Las RIAs son aplicaciones web que exhiben widgets, comportamientos y características que están presentes en las aplicaciones de escritorio. También, poseen una mayor capacidad de respuesta, son más seguras y presentan una interfaz más avanzada con respecto a las aplicaciones del modelo Web 1.0. Sus características principales incluyen, el paradigma de página única, un avanzado esquema de comunicación (con la inclusión de tecnologías push y comunicación asíncrona entre el cliente y el servidor y un manejo optimizado de los datos, reduciendo las solicitudes al servidor) y finalmente, la inclusión de un motor en el cliente (en la forma de máquina virtual o extensiones (plug-ins) en el navegador que administra la disposición gráfica de los elementos y la mayoría de las interacciones locales” [].

De las definiciones anteriores, puede notarse el hecho de que en las RIAs se busca que las aplicaciones tiendan a comportarse lo más similarmente a las aplicaciones de escritorio, evitando el refrescado excesivo de las páginas, más precisamente, permitiendo el refrescado parcial de ciertas zonas que son relevantes o que necesitan actualizarse dado un cambio de estado o debido a una actualización en la fuente de datos. También puede resaltarse que el refrescado parcial de las páginas es posible, debido a la comunicación asíncrona existente entre el cliente y el servidor, que mejora el intercambio de los datos que se transmiten por la red. Los *widgets* o elementos de interfaz interactivos son otra de las características que se encuentran presentes en las RIAs y que ofrecen una mayor riqueza a las interfaces de usuario, como así también los elementos multimedia como audio y video *streaming*. El lado del cliente en las aplicaciones RIAs, funciona de una manera más independiente del lado servidor y en ocasiones es posible utilizar las aplicaciones de manera *offline*. Finalmente, todos estos objetivos son alcanzados, agregando un motor en forma de plug-in en el cliente para la administración de las comunicaciones y para la gestión de las interacciones locales.

La definición propuesta por [], engloba la mayor cantidad de características que son comunes en las definiciones anteriormente presentadas y por ende, resulta ser la más completa. En este trabajo de fin de carrera, se tendrán en cuenta características RIAs presentes en esta definición.

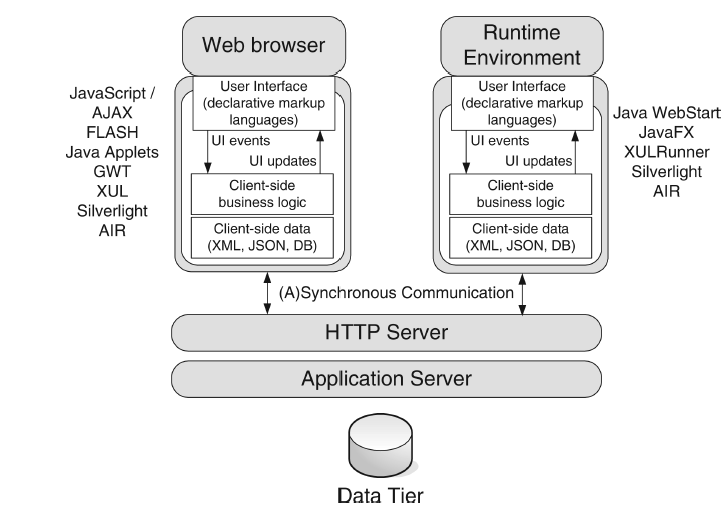


Figura 1 Arquitectura RIA

En [] se describe la arquitectura de las RIAs de manera general como se muestra en la Figura 1. El sistema está compuesto de una aplicación web servidor y un conjunto de aplicaciones de usuario corriendo en las máquinas clientes. Estas aplicaciones son implementadas o bien, en un navegador web utilizando una variedad de tecnologías como *Javascript*, animaciones Flash, código interpretado en *plug-ins* y *Java applets*, o fuera del navegador en términos de binarios descargados desde la web e interpretados en un ambiente especifico de ejecución, por ejemplo, utilizando tecnologías como Java Web Start[[11]](#footnote-11) y Adobe AIR[[12]](#footnote-12).

Las RIAs hoy en día juegan un papel preponderante. Según un estudio de mercado patrocinado por la empresa Adobe en 2011, dada las mejoras con respecto a la interfaz de usuario y al comportamiento de las aplicaciones, las RIAs han conseguido incrementar la productividad y la satisfacción de los usuarios que llevan a cabo operaciones en internet, debido, en gran medida, a la nueva experiencia de interacción que ofrecen [<rogowskimarch122007>]. Un estudio similar [<kiewe2011>], presenta datos cuantitativos con referencia a como una aplicación con características de las RIAs, puede mejorar las utilidades y disminuir los costes de desarrollo en una compañía

**2.2 Características de las RIAs**

A continuación se presentan las características más distintivas de las RIAs con respecto a las aplicaciones Web tradicionales, que fueron presentadas en [] y []. Además, por cada una de estas características, se muestra un cuadro en donde se reflejan las ventajas y desventajas de cada característica.

**2.2.1 Almacenamiento de los datos**

En las RIAs, es posible almacenar datos en el lado cliente, con diferentes niveles de persistencia (temporalmente, mientras la aplicación está en ejecución, o persistentemente). También, los datos pueden distribuirse entre ambos pares, cliente y servidor.

En la Tabla 1 se presentan algunas ventajas y desventajas de llevar a cabo una distribución de datos entre el cliente y el servidor

|  |  |
| --- | --- |
| **Ventajas** | **Desventajas** |
| * Es posible utilizar la aplicación sin necesidad de establecer una conexión con el servidor (uso offline). * Es factible la preparación y validación de los datos en el lado del cliente. | * Existe la posibilidad de replicación de datos en ambos pares. * Puede llegar a tornarse complejo establecer políticas para la asignación (distribución) de los datos. |

Tabla 1 Ventajas y desventajas de la distribución de los datos en el cliente

## 2.2.2 Lógica de negocio

En una aplicación web tradicional, la extracción de datos y la lógica de negocio se computan en el servidor. En las RIAs es posible llevar a cabo operaciones complejas directamente en el cliente (por ejemplo: efectuar navegaciones, filtrados y ordenamiento de los datos con múltiples criterios; operaciones de dominio específico para sistemas complejos; y validación local de datos). También es factible distribuir la lógica de negocios entre el cliente y el servidor para, por ejemplo, validar algunos campos de un formulario en el cliente y otros en el servidor. Por lo tanto, el diseño conceptual debe responder a la decisión de cómo asignar la computación tanto de las páginas como así también de los componentes de éstas [].

En la Tabla 2 se presentan algunas ventajas y desventajas de una computación distribuida de páginas entre el cliente y el servidor:

|  |  |
| --- | --- |
| **Ventajas** | **Desventajas** |
| * Validación de datos en vivo. * La posibilidad de utilizar la aplicación sin necesidad de una conexión (uso offline de la aplicación). | * La lógica de la aplicación en conjunto se complica. * Puede ser confuso definir si una funcionalidad en particular debe computarse en el cliente o en el servidor. * El restablecimiento de la comunicación entre el cliente y el servidor luego de la utilización offline de la aplicación es una acción propensa a errores. |

Tabla 2 Ventajas y desventajas de una computación distribuida de páginas

## 2.2.3 Comunicación entre el cliente y el servidor

Con las RIAs se crean mecanismos para reducir al mínimo la transferencia de los datos migrando las capas de interacción y presentación del servidor al cliente. Las RIAs soportan comunicaciones asíncronas entre el cliente y el servidor para la distribución de objetos de dominio, datos y la computación.

Figura 2 Comunicación síncrona versus comunicación asíncrona.

En la se Figura 2 puede ver una comparativa con respecto a la comunicación entre los pares cliente y servidor, de las aplicaciones de la web 1.0 y las actuales basadas en RIAs. En las aplicaciones web tradicionales, los datos residen en el servidor, y los clientes a medida que necesitan alguna actualización de página, llevan a cabo la solicitud de actualización por medio de la activación de algún enlace navegacional (que puede ser algún hipervínculo, botón de solicitud de registro de usuario, etc.). Seguidamente, en respuesta a la solicitud del cliente, el servidor devuelve la página con la actualización correspondiente. La comunicación es llevaba a cabo de una manera síncrona, en donde un evento del usuario es necesariamente el elemento disparador de una solicitud al servidor. Con las RIAs, un motor instalado en el cliente es el encargado de gestionar las solicitudes de transferencia de los datos al servidor y de gestionar los cambios en la disposición de los distintos elementos en la interfaz del usuario. Las solicitudes al servidor al ser gestionadas asíncronamente por un motor (o *plugin*) instalado en el cliente, permite a la aplicación llevar a cabo diversas acciones en paralelo, como por ejemplo, actualizar distintas porciones de una misma página en un momento dado. En la Tabla 3 se muestran algunas ventajas y desventajas de una comunicación asíncrona entre el cliente y el servidor:

|  |  |
| --- | --- |
| **Ventajas** | **Desventajas** |
| * Es posible llevar a cabo el refrescado parcial de las páginas, abarcando solamente las zonas de interés. * Se mejora la interacción del usuario con la aplicación. * *Server-push[[13]](#footnote-13)*. | * Se incrementan los esfuerzos de desarrollo de las aplicaciones. * El testing de las aplicaciones se vuelve más complejo. |

Tabla 3 Ventajas y desventajas de una comunicación asíncrona en entre el cliente y el servidor

## 2.2.4 Presentaciones enriquecidas

Las interfaces de usuario ofrecen una mayor riqueza con el manejo de eventos en el lado del cliente y los *widgets* interactivos que son micro programas empotrados dentro de las páginas web y administradas por un motor de *widgets* (que podría ser un plug-in instalado en el navegador). Los *widgets* presentan funciones bien específicas que por lo común resultan de utilidad a los usuarios tales como: presentar el estado del tiempo, la hora de diversos países, la cotización de las monedas extranjeras, calculadoras, entre otros. Los elementos multimedia dentro de las páginas como la intrusión de audio y video de alta calidad, a la par de animaciones también son características típicas de las RIAs, como así también, la capacidad de arrastrar y soltar elementos dentro de la interfaz, las auto-sugerencias de datos a medida que se va escribiendo un patrón en un campo y el refrescado automático de las páginas (o porciones de esta), son otras de las características interesantes que pueden encontrarse.

En la Tabla 4 se presentan algunas ventajas y desventajas de un comportamiento más sofisticado en la interfaz de usuario:

|  |  |
| --- | --- |
| Ventajas | Desventajas |
| * Funcionamiento como una aplicación de una sola página, evitando de esta forma perderse en la navegación del sitio web. * Se presenta al usuario una interfaz mucho más enriquecida y reactiva a eventos. | * Pueden presentarse problemas de rendimiento. * Es posible que se tengan incompatibilidades en el navegador web. |

Tabla 4 Ventajas y desventajas de un comportamiento más sofisticado en la interfaz de usuario

De todas las características subyacentes a las RIAs anteriormente descriptas, las presentaciones enriquecidas (que abarcan el manejo de eventos en el lado del cliente, los *widgets* interactivos, el paradigma de una sola página y el contenido multimedia) son las que representan el *look and feel* final.

## 2.3 Tecnologías para la implementación de las RIAs

Actualmente, las capacidades de las RIAs se pueden implementar en diferentes tecnologías cliente que pueden clasificarse en tres categorías principales, de acuerdo con el entorno de ejecución:

• **Basadas en *Javascript*:** la lógica del lado cliente está implementada en *Javascript* (el enfoque también es conocido como "Ajax", *Asynchronous Javascript y XML* [] y las interfaces de usuario se basan en una combinación de HTML y CSS.

Figura 3 A la izquierda el Modelo de aplicación web clásico. A la derecha, el modelo de aplicación web Ajax.

La principal ventaja de este enfoque es que se basa en el *Javascript* incorporado en el navegador y soporta los estándares de W3C. En la Figura 3 se presenta el modelo de aplicación Ajax en comparación con el modelo de aplicación web clásico. Como puede apreciarse para el caso del modelo Ajax, el motor Ajax es el encargado de orquestar la disposición de los elementos en la interfaz de usuario en el lado del cliente por medio de HTML y CSS y la lógica de este por medio de *Javascript*. Por lo general la comunicación con el lado servidor es llevado a cabo por medio de solicitudes HTTP o HTTPS y la respuesta del lado servidor se da por medio de lenguajes de marcado como XML o JSON. Los principales inconvenientes son el soporte multimedia insuficiente, limitaciones en las cajas de arena (sandboxes) del navegador, por ejemplo, el acceso al sistema de archivos o almacenamiento persistente, y la inconsistencia en el comportamiento del navegador. Debido a este último aspecto, un gran número de bibliotecas se han propuesto para permitir a los desarrolladores abstraerse de las idiosincrasias del navegador.

• **Basadas en *plug-ins*:** en donde la representación avanzada y el procesamiento de eventos se encomienda a los *plug-ins* del navegador por medio de la interpretación de lenguajes específicos de scripting, XML o archivos multimedia. Una ventaja de los *plug-ins* es que generalmente soportan la interacción multimedia de forma nativa, permitiendo la persistencia en el lado del cliente y ofrecen un mejor desempeño que *Javascript* interpretado. Algunos *plug-ins* vienen ya instalados en los navegadores, pero otros requieren de la intervención del usuario administrativo. Sin embargo, en algunos casos no proveen el acceso a servicios del sistema operativo (por ejemplo, al sistema de archivos).

• **Basadas en ambientes en tiempos de ejecución:** las aplicaciones se descargan de la Web, pero se ejecutan fuera del navegador, utilizando un ambiente de escritorio en tiempo de ejecución. Estas soluciones ofrecen lo máximo en términos de capacidades de cliente y el uso off-line, con pleno acceso al sistema operativo subyacente. Sin embargo, se basan en un ambiente especializado en tiempo de ejecución, lo que obliga a los usuarios a que lo instalen (y podría no estar disponible en todas las plataformas, como por ejemplo en teléfonos móviles). Muchas de las tecnologías RIAs se pueden utilizar para desarrollar aplicaciones de este tipo. En la Tabla 5 puede apreciarse las capacidades y las limitaciones de cada una de las tecnologías con respecto a las características descritas en la sección 2.2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tecnología cliente vs  Características de las RIAs | Presentaciones enriquecidas | Almacenamiento de los datos en el cliente | Lógica de negocio en el cliente (o distribuida entre el cliente y el servidor) | Comunicación entre el cliente y servidor |
| Basados en JavaScript | Limitada: sin multimedia | Limitada: no hay persistencia de datos | Si | Si |
| Basados en Plug-ins | Si | Si : con *plug-ins* adicionales | Si | Si |
| Basados en ambientes en tiempo de ejecución | Si | Si | Si | Si |

Tabla 5 Tecnología cliente vs Características de las RIAs

Entre las tecnologías citadas, la práctica más popular de desarrollo RIA ve la adopción de las basadas en *Javascript* (es decir, AJAX) como la opción más común. Las limitaciones actuales se suelen resolver utilizando extensiones de Flash para el procesamiento de vídeo (y *Google Gears* o *Flash Shared Objects* cuando en el lado del cliente el almacenamiento persistente es necesario). Las razones principales que explican esta tendencia son en el hecho de que: AJAX es asumido por muchos desarrolladores como el conjunto más abierto y estándar de tecnologías (y más cerca de la especificación HTML5) y que no requiere acciones administrativas (por ejemplo, la instalación de software) de los usuarios. También se pueden combinar fácilmente con *plug-ins* que se construyen para superar sus limitaciones.

**2.4 Herramientas para el desarrollo de las RIAs**

En la sección anterior se presentaron las diferentes tecnologías para el desarrollo e implementación de las RIAs. Para este trabajo de fin de carrera, se optó por herramientas y frameworks de desarrollo de uso abierto y que son de amplia utilización en la comunidad web. Se ha señalado el hecho de que las implementaciones basadas en *Javascript* o librerías Ajax son las más utilizadas en la actualidad, debido a que utiliza tecnologías de uso abierto estandarizado como lo son; *Javascript*, HTML y CSS. También esta forma de implementar las RIAs es el más cercano al estándar HTML5. RIAs

Son numerosas las librerías *Javascript* existentes en la actualidad. Estas librerías tienen como objetivo abstraer a los desarrolladores de tener que lidiar directamente con el DOM (*Document Object Model*) para la disposición de los elementos en las páginas web, ofreciendo capas de software amigable, reduciendo notablemente los tiempos de desarrollo y mejorando la productividad. En la Figura 4 se puede apreciar algunas librerías *Javascript* de uso extendido



Figura 4 Algunas librerías *Javascript* de uso común

Estas librerías también buscan explotar el lado del cliente en las aplicaciones y minimizar las interacciones con el lado servidor, para que de esta forma se obtenga un mejor rendimiento. A la par de permitir a los desarrolladores implementar aplicaciones a un alto nivel de abstracción, las librerías ofrecen una gran variedad de *widgets* interactivos que son de uso común en las aplicaciones web.

Los *widgets* para las RIAs representan elementos enriquecidos para la interfaz de usuario, que tienen como objetivo ofrecer una mayor interactividad, dada sus características dinámicas y un comportamiento general, similar a los patrones de comportamiento. Los widgets son microprogramas que cumplen una función predeterminada y que a la vez sus propiedades pueden ser modificadas para expresar comportamientos personalizados por el usuario. Una vez modificada las propiedades del widget, éste es introducido dentro de la aplicación para cumplir una función en particular.

**2.4.1 Widgets más utilizados**

En estudios llevados a cabo en 2009[[14]](#footnote-14) y 2010[[15]](#footnote-15) , se presenta un análisis de los *widgets* más utilizados por las aplicaciones web. En la Figura 5 se presentan de manera sumarizada tales *widgets*. De todos los *widgets* identificados, suena interesante determinar cuáles son los más utilizados hoy en día, por lo que se ha realizado un análisis de portales web populares (Facebook, Gmail, Youtube y Amazon) para determinar que *widgets* son comunes en estos sitios. El análisis determinó que los cuatro portales utilizan los siguientes *widgets*;

***Accordion:*** Muestra paneles de contenido plegable para presentar la información en una cantidad limitada de espacio.

***Tabs:*** Ofrece una sola área de contenido con múltiples paneles, cada uno asociado con una cabecera en una lista.

***Autocomplete:*** Permite a los usuarios a seleccionar y encontrar rápidamente de una lista previamente poblada de valores a medida que se escribe, mejorando la búsqueda y el filtrado.

***Tooltip:*** Ofrece mensajes personalizados de sugerencia sobre los elementos de interfaz, reemplazando los mensajes nativos.

Figura 5 Elementos de interfaz de usuario enriquecidos (*widgets*) más utilizados.

***Datepicker:*** Permite seleccione una fecha de un calendario emergente o *inline*.

***Live validation:*** Ofrece validaciones en vivo de los campos en los formularios.

**2.4.2 Las librerías *Javascript* *jQuery*UI[[16]](#footnote-16) y *jQuery Validation Plugin*[[17]](#footnote-17)**

Estudios de mercado recientes han presentado a *jQuery*[[18]](#footnote-18) como la librería *Javascript* más utilizada a nivel global[[19]](#footnote-19). *jQuery* es de uso abierto y ha tenido un crecimiento notable en términos de evolución hasta hoy en día desde su aparición en el año 2005. Esta librería a la par de presentar una manera robusta y confiable para desarrollar código *Javascript*, en su versión *jQueryUI* se encuentran numerosos *widgets* interactivos idóneos para las diversas interfaces de usuario, como para aplicaciones web y móviles.

De todas características citadas en la sección anterior, solamente el *Live validation* no es soportado por *jQueryUI* de manera nativa. Sin embargo con *jQuery Validation Plugin*, que es una extensión basada en *jQuery*, es factible llevar a cabo validaciones enriquecidas sobre los campos de un formulario de una manera bastante intuitiva.

Con *jQueryUI* y *JQuery Validation Plugin*, es posible dar cobertura a todos los widgets que serán tenidos en cuenta en este trabajo de fin de carrera.

**2.5 Síntesis del Capítulo**

En este capítulo se han visto las diversas características que ofrecen las RIAs, como así también los enfoques tecnológicos para explotar el lado del cliente en este tipo de aplicaciones. Estos enfoques son: las implementaciones basadas en librerías *Javascript,* las basadas en la instalación de *plug-ins* o las que se enfocan en ambientes en tiempo de ejecución. La primera de ellas es la más utilizada en la comunidad web, debido a que la aplicación web se implementa por medio de un compendio de estándares abiertos trabajando conjuntamente como lo son HTML y CSS *(para la representación de los elementos y el posicionamiento)*, *Javascript (para la lógica de la aplicación en el lado cliente)* y *XML* o JSON*(para la comunicación entre el cliente y el servidor)*.

Son varias las librerías *Javascript* existentes en la actualidad[[20]](#footnote-20). Algunas de ellas permiten la representación de ciertos elementos de interfaz interactivos (*widgets*) que son comunes en las interfaces enriquecidas actuales y a la vez ofrecen la posibilidad de agregar cierta lógica en el lado cliente, como validaciones locales de campos de entrada en los formularios. Según el análisis llevado a cabo a principales portales web, dentro de los *widgets* más utilizados se ven a los *tooltips*, *tabs, accordion, datepicker y autocomplete,* como así también diversas validaciones locales en los campos de entrada como validaciones de tipo numérico, email, password etc. *jQuery* a la par de ser la librería *Javascript* más popular actualmente, ofrece en sus versiones *jQueryUI* y *JQuery Validation Plugin,* cobertura a todas estas características enriquecidas.

**CAPITULO 3**

**ESTADO DEL ARTE DE LAS METODOLOGÍAS WEB BASADA EN MODELOS PARA LAS RIAS**

En el capitulo anterior se presentó una visión general de las RIAs, con sus características principales y las diferentes tecnologías utilizadas para el desarrollo de las mismas. Se ha 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 los basados 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 basadas en modelos, presentando primeramente los conceptos de MDSE (*Model Driven Software Engineering*), MDD(*Model Driven Development*) y MDA (*Model Driven Architecture*) para posteriormente dar píe a las metodologías web existentes basadas en modelos que presentan características de las RIAs. Finalmente se presentará a la metodología web MoWebA (*Model Oriented Web Aproach*), una metodología web separada en capas, que sigue el paradigma MDA) para el ciclo de desarrollo de sus aplicaciones, que resulta prometedora para la implementación de características de las RIAs.

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

Los modelos son de suma importancia para entender y compartir conocimiento acerca de un software complejo. MDSE 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 MDSE, los conceptos principales son los modelos y las transformaciones (esto es, manipulaciones y/o operaciones sobre los modelos).

MDSE, 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 MDSE 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 MDSE, es posible adoptar un enfoque MDDpara 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[[21]](#footnote-21) es un estándar impulsado por el consorcio OMG (*Object Management Group*) que contiene en si misma a varios estándares de facto, tales como UML[[22]](#footnote-22) *(Unified Modeling Language*) , OCL[[23]](#footnote-23) (*Object Constraint Language*), MOF[[24]](#footnote-24)(*Meta Object Facility*), *QVT[[25]](#footnote-25)*(*Query View Transformation*), entre otros; con la meta presente de promover el desarrollo de software para diversos dominios de aplicación, como los son las aplicaciones para el ámbito de las finanzas , las telecomunicaciones , las aplicaciones aeroespaciales, las embedidas, 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 Figura 1. Los meta-metamodelos (M3) se expresan por medio MOF o ECORE para el *Eclipse Modelling Framework* (*EMF*) y los metamodelos (M2) de la aplicación se expresan por medio de un *General Purpose Modelling Lenguage (GPML)* que por lo general es 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 este caso un video, se representa en M0.



Figura 1 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 . La fase *Computation independent Model* (CIM) corresponde a los documentos, modelos o diagramas utilizados para la toma de requerimientos en una aplicación en particular, independientemente de cómo han sido implementados. Representan al 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 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 un lenguajes de transformación como QVT o ATL. 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 por herramientas de trasnformación M2T como MOFScript, Acceleo u JET (*Java Emmitter Template*).

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

Uno de los aportes de MDSE, 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 MDSE 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, la generación de código es el proceso de transformar modelos en código fuente.

Dentro de las preguntas esenciales cuando uno tiene que desarrollar un generador de código basado en modelos se encuentran las siguientes:

**¿Qué tanto código va a generarse?**

La pregunta principal aquí es que partes 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 tiene que ser implementadas manualmente.

**¿Qué código va a generarse?**

Implica que 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, capaz de representar un sistema de la mejor manera.

**¿Como a va generarse?**

Muchos lenguajes pueden ser empleados para generar código a partir de los modelos y pueden ser GPLs y DSL.

**3.3 Beneficios de los lenguajes de transformación (M2T)**

**Separación de código estático y dinámico**:

Los lenguajes de transformación M2T separan el código estático y dinámico, utilizando el enfoque de plantillas (*templates)* 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 simple para las partes estáticas y los llamados metamarcadores (*meta-markers)* para las partes dinámicas. Los metamarcadores son marcadores de posición y deben ser interpretadas por un motor de *templates* que procesa los *templates* 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 *templates.*

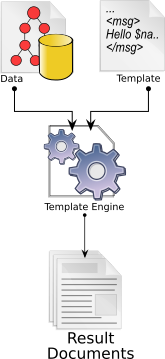


Figura 3 Plantilla, motor de plantillas y modelos de entrada para producir texto

Los *templates* 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.

Dentro de los metamarcadores, el código es utilizado para acceder a la información almacenada dentro de los modelos.  El estándar OCL es la elección para llevar a cabo esta tarea en la mayoría de los lenguajes de transformación basados en *templates*. 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.

**3.4 UNA VISTA DE LOS LENGUAJES DE TRANSFORMACIÓN BASADOS EN TEMPLATES**

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

**XSLT[[26]](#footnote-26)**

La serialización XMI de los módulos, 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 serializaición XMI, que requiere ciertos conocimientos adicionales de como los modelos son actualmente codificados como archivos XML. Así, el enfoque opera directamente a nivel de modelo.

**JET Java Emitter Template[[27]](#footnote-27)**

Fue uno de los primeros enfoques para 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 *templates*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[[28]](#footnote-28)**

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 a este proyecto se llama Xtend, que está basado en Java,pero ofrece muchas características adicionales propias del lenguaje. Por ejemplo, es posible incrustar *templates* de generación de código (para tener una sintaxis similar  al template Xpand) dentro del código Xtend.

**MOFScript[[29]](#footnote-29)**

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

**Acceleo[[30]](#footnote-30)**

*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 (MTL). 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.

**3.5 Principales enfoques de desarrollo web basado en modelos para las RIAs**

En [] y [], se identifica la necesidad de metodologías sistemáticas para el desarrollo de las RIAs y se llevan a cabo estudios presentando las diversas metodologías web existentes para ese fin. El 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 en las que se incluyen a WebML-RIA[], OOHDM -RIA[], OOH4RIA[] , (UWE-R[]Patrones con UWE[] y UWE+RUX []).
2. Enfoques de desarrollo sistemáticos provenientes de la comunidad *Human Computer Interactión (HCI),* en donde el diseño RIA es el foco de la metodología RUX[<trigueros2007>] y puede ser logrado de igual forma con el enfoque más general UsiXML[] [].
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 se analizan las metodologías web anteriores, una de las consideraciones a tener en cuenta, es que las mismas adopten estándares (por ejemplo, UML). También se busca que las metodologías en cuestión, sean de uso abierto para la comunidad de desarrolladores y no propietarias. He ahí que a continuación se describirán brevemente las metodologías basadas en UML que son OOH4RIA, UWE-R, Patrones con UWE, UWE combinado con la herramienta RUX y los patrones de interacción con OOWS. El resto de las metodologías se presentarán en el cuadro comparativo con sus respectivos alcances para las RIAs. La categoría d) del estudio mencionado no se considera debido a que son propuestas cerradas basadas en herramientas comerciales.

**a1) 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 RIAs, como así también, los actores que participan en el ciclo de desarrollo. OOH-RIA, 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[[31]](#footnote-31): este metamodelo permite la especificación de los aspectos estructurales de las RIAs. 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 par el lado servidor.

Figura 4 Representación del proceso MDD para OOH-RIA

**a2) Extensiónes 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.

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



Figura 6 Extensiones al metamodelo de Presentación en UWE-R.

Por último, se llevan a cabo extensiones con respecto al proceso (o la lógica de la aplicación), en la cual, se puede 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 en el caso que expire un temporizador; se dispara automáticamente alguna acción.

**a3) 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 extiende 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 RIAs, 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 específica para un dispositivo o un grupo de dispositivos y para una plataforma de desarrollo RIA tal como Flex, Ajax o OpenLaszlo[[32]](#footnote-32).

**a4) UWE con patrones**

UWE puede extenderse por medio de patrones[].Los patrones RIAs, 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.

Figura 7 Patron *Periodic Refresh* en UWE

Figura 8 Patron Autocomplete en UWE

Este enfoque consiste en el uso de modelos que representan *widgets* RIAs, 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 las y se muestran los patrones definidos en propuesta sugerida a la metodología UWE.

**a5) Patrones en OOWS**

La principal contribución de este trabajo es un modelo de interacción para especificar la nueva semántica para hacer frente al desarrollo basado en modelos RIAs. [<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*)[[33]](#footnote-33)*, 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 se puede analizar el proceso de desarrollo para esta propuesta



Figura 9 Un resumen del enfoque MDD con patrones para OOWS de Valverde y Pastor

La Tabla 7 que se muestra a continuación presenta un resumen de las propuestas, indicando si las características RIAs contempladas por las mismas.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 6, notamos que la metodología que más características de las RIAs abarca es WebML con la salvedad que utiliza herramientas propietarias para su modelado[[34]](#footnote-34), 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 RIAs 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.

**3.6 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 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 entre los que se citan:

1.  **Modelado del problema:** en el que se incluyen 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. De aquí es posible llevar a cabo transformaciones para obtener los modelos específicos de la plataforma de manera semi-automática por medio de reglas.

2.   **Modelado de la solución**: en donde forman parte el ASM (*Architectural Specific Model*) y el 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 RIAs, debido a que existen numerosas plataformas destino para desplegarlas. En las aproximaciones estudiadas, por lo general las extensiones RIAs son definidas en el marco de los modelos conceptuales (PIMs), haciendo que los modelos que deberían ser independientes de la solución, adquieran elementos que ya son propios de una arquitectura específica.



Figura 10 Niveles y fases 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, no contemplados en otras metodologías. Esto hace que al definir los modelos propios de las RIAs, 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.

**3.6.1 La capa de presentación de MoWebA**

La capa de presentación de MoWeba abarca a los metamodelos de contenido y estructura. En el metamodelo de contenido (*Content*), se tienen los diversos elementos de interfaz (*uIElements*) correspondientes a la web 1.0, en la que se tienen a los *textImput*, a los enlaces que podrían corresponder a navegaciones internas de la aplicación (*anchor*) o bien a navegaciones externas (*externalLink*), a los *button,* a los elementos del tipo selección que corresponde a los *choice*  y a los *dropBox*, a los *text* para texto plano en las páginas, al *htmlText* para el despliegue de cualquier texto HTML, y al elemento del tipo *multimedia* para audio y video. Cada uno de estos elementos cuenta con sus respectivos *atributos* para identificar a sus propiedades intrínsecas como puede apreciarse en la . 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 *uIElements* dentro 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.

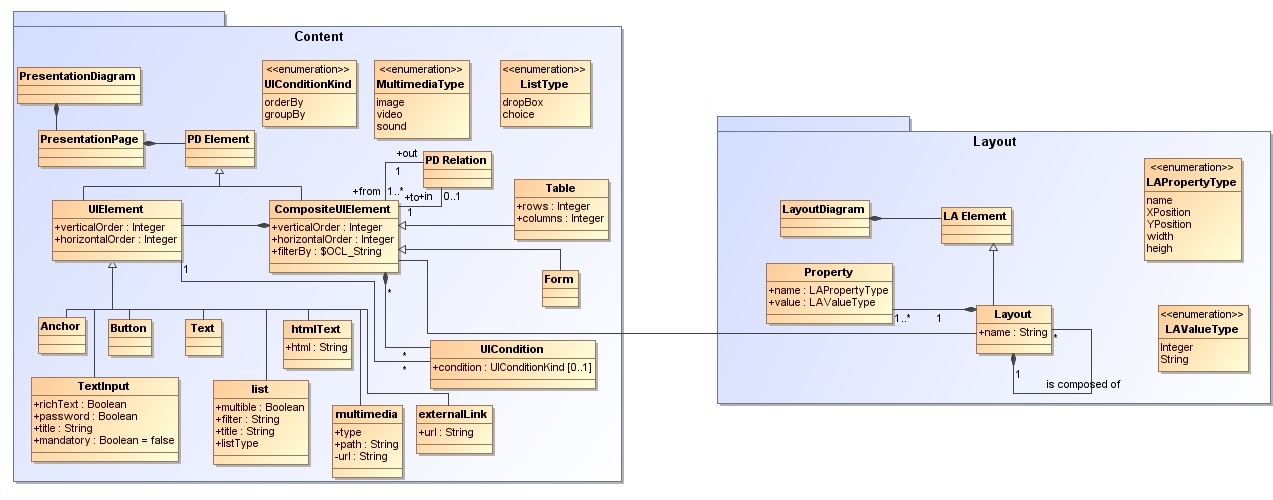


Figura 11 Metamodelo de Contenido y Estructura de MoWebA

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 y en la .

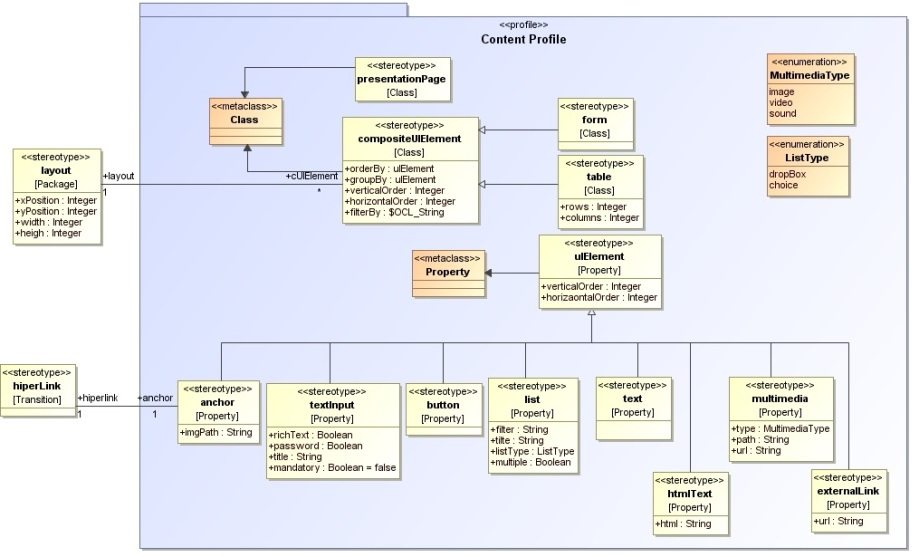


Figura 12 Perfile de Contenido de MoWebA

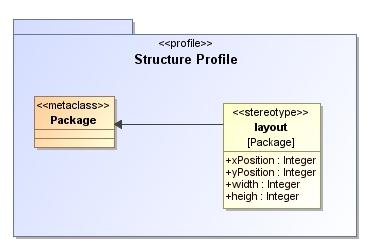


Figura 13 Perfil de estructura de MoWebA

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

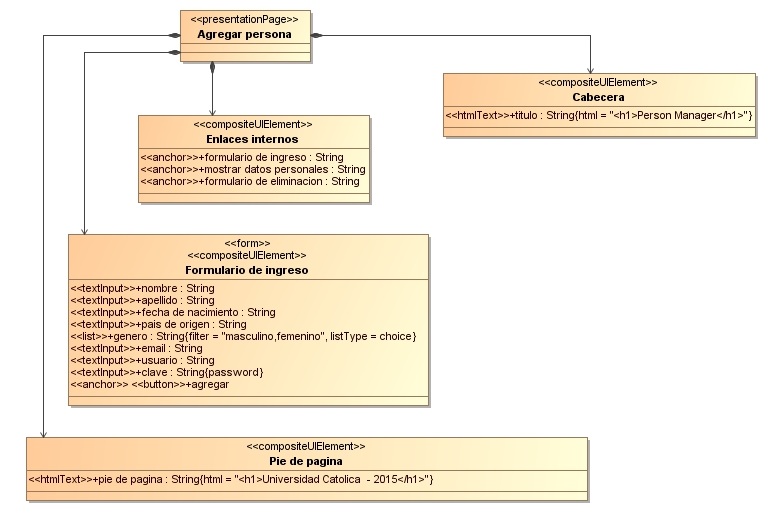


Figura 14 PIM modelado con el perfil de contenido de MoWebA

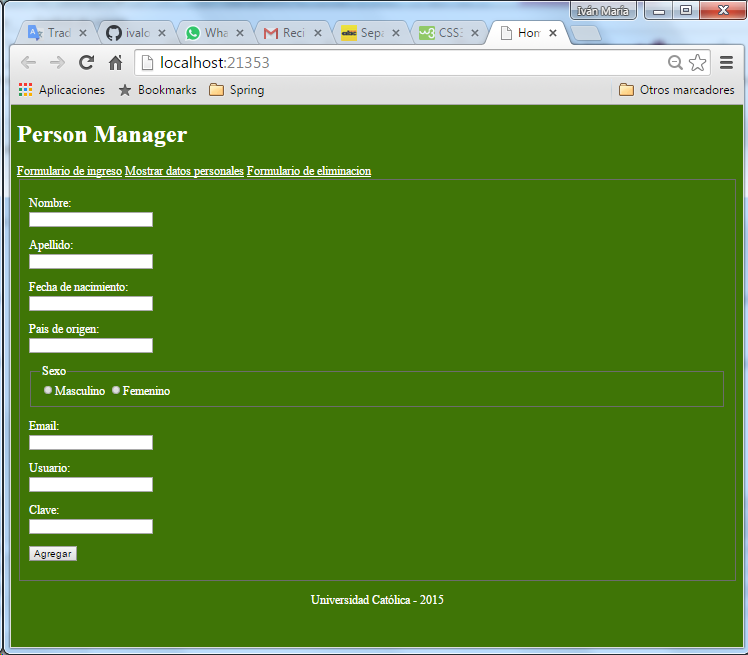


Figura 15 Interfaz obtenida a partir del PIM presentado en la Figura 13

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

La representa el proceso para el modelado y generación de interfaces. 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[[35]](#footnote-35). Posteriormente tanto los PIM como los perfiles son exportados al formato xmi del EMF[[36]](#footnote-36). Esto de por sí es llevado a cabo a fines de tener compatibilidad con la herramienta de transformación M2T Acceleo[[37]](#footnote-37), que toma como entrada modelos UML que están basados en el metamodelo Ecore[[38]](#footnote-38).

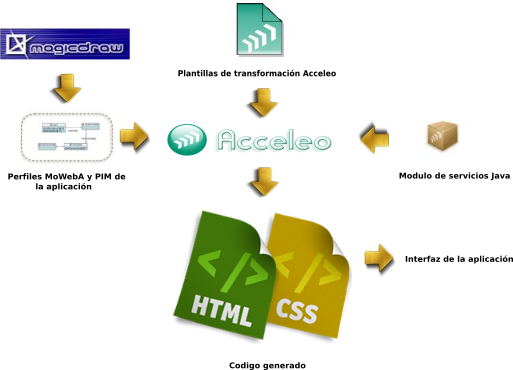
Una vez exportados los modelos (PIM y profile) al Acceleo, por medio de las plantillas de transformación y los módulos de servicio en Java (*Java Service Wrappers*), que forman parte de Acceleo, es posible llevar a cabo las transformaciones necesarias sobre los modelos de entrada para obtener los archivos 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 16 Fases para la generación de interfaces con MoWebA

**3.7 Síntesis del Capítulo**

Los modelos hoy en día ocupan un lugar importante en proceso de desarrollo de software, comúnmente para la comunicación entre los desarrolladores y las personas sin conocimientos técnicos u bien entre los mismos desarrolladores. Las metodologías de desarrollos 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 metodologías 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 RIAs 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 RIAs, 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.

**Capítulo 4**

**Una Extensión RIA para la APROXIMACION web MoWebA**

Se ha visto en el capítulo anterior, una breve introducción de los alcances de la metodología web MoWebA, presentando sus diferentes capas y fases de desarrollo y transformación. Se ha mencionado el hecho de que MoWebA resulta ser una metodología flexible para llevar a cabo extensiones que le permiten, de cierto modo, mantenerse vigente con los nuevos avances que constantemente afectan a las aplicaciones web. También se ha tenido en cuenta el hecho de que las RIAs forman parte de esa evolución y que las metodologías web basadas en MDD/MDA necesitan tener en cuenta estos cambios.

Entre las diversas características que presentan las RIAs, las presentaciones enriquecidas toman un papel preponderante debido a que proveen el dinamismo e interactividad que las diferencia de las aplicaciones de la web 1.0. Los widgets interactivos colaboran de manera notable a este enriquecimiento, y tanto es así que en la actualidad es difícil encontrar aplicaciones web que carezcan de estos elementos para la interfaz de usuario.

Sin embargo, se ha visto que las diversas metodologías presentadas basadas en MDD/MDA ofrecen cierta cobertura con respecto a los diversos tipos de widgets RIAs existentes, pero o bien los mecanismos de extensión para la cobertura son muy tediosos, con numerosas cadenas de transformaciones M2M y M2T (como en el caso de OOH4RIA), o bien las herramientas para llevar a cabo el enriquecimiento son de uso propietario (como en los casos de UWE+ RUX). También se ha notado que muchas de las transformaciones M2T no se llevan a cabo automáticamente sino de manera semiautomática o manual, como es el caso de UWE con patrones.

En su definición original, la capa de presentación de MoWebA contiene diversos elementos para la interfaz de usuario que son de uso común en la aplicaciones web 1.0. En este capítulo se presentarán nuevos elementos que forman parte de la extensión propuesta, precisamente los widgets comunes en las RIA que fueron presentados en la sección anterior. Los nombres de tales *widgets* (*accordion*, *tabs*, *autocomplete*, *datePicker*, *tooltip* y el *fieldLiveValidation*) serán presentados en MoWebA como *richAccordion*, *richTabs*, *richAutoSuggest,* *richDatePicker*, *richToolTip* y *richFieldLiveValidation* respectivamente.

y una breve discusión acerca del enfoque tomado para el modelado de widgets con MoWebA.

**3.1 El enfoque utilizado con MoWebA para la generación de interfaces enriquecidas**

La Figura 1 representa el proceso tenido en cuenta en este trabajo de fin de carrera para el modelado y generación de interfaces enriquecidas (también conocidos como los *front-ends* de las aplicaciones). Como puede apreciarse, las fases de desarrollo son similares a las presentadas en el capitulo anterior. Sin embargo, el perfil de contenido de MoWebA, ha sido extendido con nuevos elementos de interfaz de usuario que son típicos de las RIAs. También las plantillas de transformación, han sido adaptadas para generar el código correspondiente a cada uno de los nuevos elementos de interfaz RIAs que han sido agregados.

A diferencia del enfoque de presentación de MoWebA en su forma original, que genera código HTML para los elementos de interfaz de la web 1.0 y CSS para la estructura de cada uno de los elementos dentro de las páginas, en MoWebA extendido, se genera código HTML y *Javascript* para la plataforma *jQueryUI,* específicamente el código para los *widgets* *RichAccordion*, *RichTabs*, *RichDatePicker*, *RichTooltip*, y *RichAutoSuggest* y *jQuery Validation plug-in* para los dieversos tipos de validación local. De igual manera, que en su forma original, es posible generasr el código CSS para estructurar cada uno de los elementos de interfaz enriquecidos (o no) ( Finalmente las librerías *Javascript jQueryUI* y *jQuery Validation Plugin* se invocan desde el código fuente generado para tener todas las funcionalidades enriquecidas de la aplicación a partir del código generado.



Figura 1 Fases de desarrollo para la propuesta de extensión a MoWebA

**3.2 – Extensiones a la capa de presentación de MoWebA.**

El objetivo de este trabajo de tesis es agregar nuevos elementos al metamodelo de contenido, precisamente elementos de las RIA, que son los widgets interactivos y la validación en el lado del cliente en los formularios. Estos nuevos elementos serán modelados en primera instancia y luego traducidos a código por medio de una transformación M2T. Las extensiones se llevarán a cabo en el metamodelo de contenido para obtener la nueva representación de la sintaxis abstracta. De manera correspondiente también se extenderá el perfil que permitirá el modelado de la sintaxis concreta, precisamente los diversos modelos PIM representados con diagramas UML.

**3.2.1 - El metamodelo de contenido extendido para moweba .**

El metamodelode contenido extendido junto al metamodelos de estructura de MoWeba están representados en la Figura 2. En ellos se presentan los diversos elementos que permiten representar una interfaz de usuario enriquecida

Como un nuevo aporte al metamodelo de contenido de MoWebA, se propuso la clasificación de los diferentes elementos de interfaz, en elementos de entrada, salida y control respectivamente. Esto fue necesario para establecer un orden dentro de los elementos de interfaz y para una mayor claridad dentro del metamodelo de Contenido.Los distintos elementos de interfaz de usuario se clasifican en:

* ***Elementos de salida* (*OutputElements*): Comprende a los elementos de interfaz enriquecidos y tradicionales encargados de desplegar o mostrar información en las páginas de presentación. En esta categoría** se engloba **a los elementos** *text*, *htmlText* *ultimedia* y *richToolTip*
* **Elementos de entrada (*imputElements*):** Comprende a los elementos de interfaz enriquecidos y tradicionales encargadas de obtener una entrada desde la interfaz de usuario. En esta categoria se engloba a los elementos *textInputs*, *list*, *richAutoSuggest*, *richDatePicker* y *richFieldLiveValidation*).
* ***Elementos de control* (*controlElements*)*:***  Comprende a los elementos de interfaz tradicionales encargados de obtener una orden de navegación o cambio de página. En esta categoría se engloba a los elementos *externalLink*, *anchor* y *button*.

Como parte de las extensiones llevadas a cabo a MoWebA, también se encuentran los richAccordion y los richTabs, que pueden tener uno o varios paneles (*Panels).* Cada uno de estos *Panels*, permite aglomerar a muchos elementos de interfaz *compositeUIElement.* El atributo *defaultPanelNumber*, permite establecer cuál de los *Panels* existentes en un *richAccordion* o *richTabs*, se desplegará primeramente por omisión.

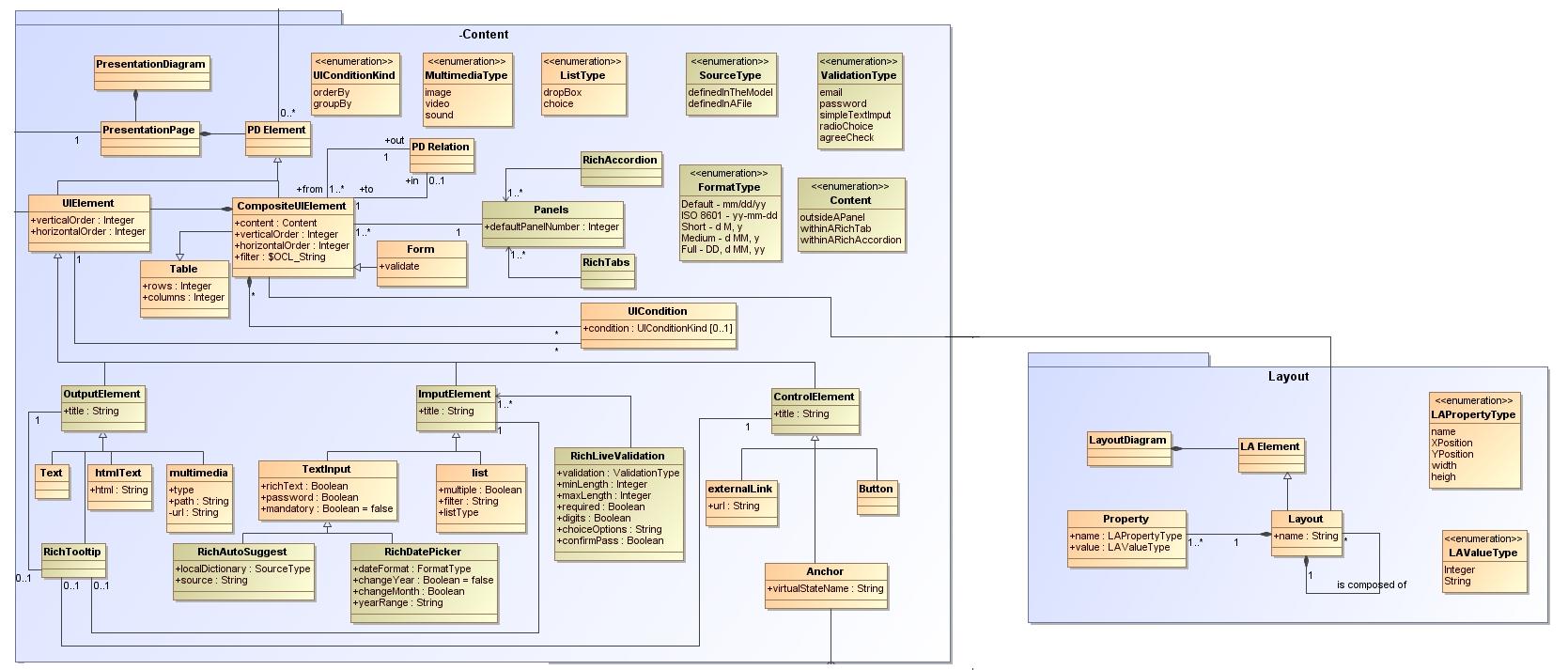


Figura 2 Metamodelo de contenido y estructura

A continuación se describen cada uno de los elementos que forman parte de la extensión al metamodelo de contenido de MoWebA.

**3.3.1 RichAutoSuggest**

Este elemento de interfaz enriquecido, contiene los valores etiquetados *localDictionary* y *source*. El primero corresponde a una enumeración que permite definir al diccionario de sugerencias, que consiste en una lista en la cual están definidas todas las palabras sugeridas del widget *richAutoSueggest*, en un archivo en formato .*xml* o bien definir el listado de sugerencias en el mismo diagrama de clases en un formato de cadenas de sugerencias separado por comas. El valor etiquetado *source* será completado de acuerdo a lo definido en la enumeración anterior. En el caso de estar definido con el valor *definedInTheModel*, entonces *source* que es del tipo *string*, contendrá un conjunto de palabras correspondiente a las sugerencias separadas entre comas. Si en la enumeración está definido el valor *definedInAFile*, entonces el diccionario de palabras está definido en un archivo xml y en valor etiquetado *source* se colocará la ruta en la cual se encuentra el archivo de palabras.

Figura 3 El *widget* RichAutoSuggest

**3.3.2 RichDatePicker**

Este elemento de interfaz enriquecido contiene a los valores etiquetados *dateFormat*, *changeYear*, *changeMonth* y *yearRange*. El *dateFormat* corresponde a una enumeración que contiene cinco formatos de fecha distintos que pueden ser:

Figura 4 El *widget RichDatePicker*

\**Default - mm/dd/yy* por ejemplo 06/08/2015

\* *ISO 8601 - yy-mm-dd* por ejemplo 2015-06-08

\* *Short - d M, y* por ejemplo 8 Jun, 15

\**Medium - d MM, y por ejemplo 8 June, 15*

\* *Full - DD, d MM, yy por ejemplo Monday, 8 June, 2015*. El valor etiquetado *changeYear* es un valor booleano que indica la ausencia o presencia de un rango de años desplegable en una lista que formará parte del *richdatePicker.* Por omisión, si *changeYear* está configurado en verdadero, se mostrará en el *datePIcker* una lista desplegable presentando los diez años anteriores a partir de la fecha actual. También es posible asignar al valor etiquetado *yearRange* un rango de años para el *richDatePicker* que se define en el formato yyyy:yyyy; por ejemplo 1970:2015. Definir *yearRange* resulta ideal para la selección de fechas pasadas, como el año de nacimiento o fechas históricas.

Por último, el valor etiquetado booleano *changeMonth*  permite desplegar una lista con todos los meses del año para una rápida selección.

**3.3.3 RichToolTip**

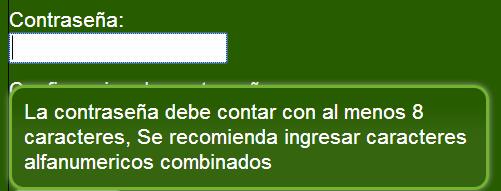
Una particularidad ocurre con el elemento richToolTip con respecto a la clasificación anterior. Este elemento de salida, tiene como objetivo enriquecer con mensajes personalizados a cualquiera de los elementos que forman parte de la clasificación de elementos de entrada, salida y control.

Figura 5 El *widget RichToolTip*

Este elemento de interfaz enriquecido no cuenta con valores etiquetados. Al definirse este estereotipo en conjunción con algunos de los elementos simples de entrada, salida o de control, implica que un mensaje emergente será desplegado cuando el puntero del mouse se posicione sobre el elemento. Cada uno de los elementos de entrada, salida y control posee el valor etiquetado *title*, que corresponde al mensaje que será desplegado

**3.3.4 RichFieldLiveValidation**

Este elemento de interfaz enriquecido, permite llevar a cabo validaciones locales a diversos elementos pertenecientes a un formulario. El valor etiquetado *validation* corresponde a una enumeración que contiene todas las validaciones que son posibles de llevar a cabo. Estas validaciones pueden ser del tipo *email*, *password*, *simpleTextInput*, *radioChoice* y *agreeCheck*. Para las validaciones del tipo *email*, *password* y *simpleTextInput* es posible establecer la cantidad mínima de caracteres en un campo de entrada estableciendo el valor etiquetado *minLength*. Similarmente el valor etiquetado *maxLength* permite establecer la cantidad máxima de caracteres que es posible ingresar en un campo de entrada. Para el *simpleTextInput,*  el valor etiquetado *digits* establece que el campo de entrada debe tener estrictamente valores del cero al nueve. Las validaciones *email*, *password* y *simpleTextInput*, pueden configurarse como mandatorias, lo que implica que no pueden quedar vacías, estableciendo el valor etiquetado booleano *required* como verdadero.

Al seleccionar el tipo de validación *password*, es posible establecer el valor booleano del valor etiquetado *confirmPass* en verdadero, para el caso en el que se necesite crear otro campo de entrada para la confirmación de contraseña.

Por otro lado, la validación del tipo *radioChoice*, permite establecer de un listado de opciones de selección, un valor obligatorio, que es establecido en el valor etiquetado *choiceOptions*, como cadenas separadas por comas. De todas las opciones existentes en *choiceOptions*, una de ellas se establece con la palabra reservada *Required*, separada con un espacio en blanco para indicar que esa opción por omisión es obligatoria y debe seleccionarse al menos esa opción. De igual forma, si se elige otra opción que no tenga la palabra reservada *Required* establecida, el campo será validado.

Finalmente, se encuentra el tipo de validación *agreeCheck*, que una vez seleccionado, permite desplegar un cuadro de selección del tipo *radio*, el cual es obligatorio y debe ser seleccionado en un formulario para que este pueda ser validado. Este campo es idóneo para acuerdos de conformidad.

El *richFieldLiveValidation* es activado en el momento en el que se establece como verdaderoel valor etiquetado booleano *valídate* dentro del elemento de interfaz *form*,que indica si el formulario en cuestión tendrá o no algún tipo de validación de campo.

**3.3.6 RichAccordion**

Este elemento de interfaz enriquecido permite encapsular a varios elementos de interfaz dentro de paneles colapsables. Dentro de los elementos que pueden ser desplegados se encuentran los *compositeUIElements*, *table* y los *form*. Cada uno de los paneles que componen el RichAccordion es representado por medio de un diagrama de clases, que contiene atributos o propiedades que representan los distintos elementos de interfaz y valores etiquetados para indicar que el panel contendrá elementos compuestos.

Para establecer que un panelforma parte de un *richAccordion*, se debe seleccionar de la enumeración *content* que se encuentra dentro de un *compositeUIElement*, el valor *withinARichAccordion*. El valor por omisión establecido para la enumeración *content* es *independent* que especifica que el contenido no forma parte de un panel.

**3.3.7 RichTabs**

Este elemento de interfaz enriquecido, permite al igual que el *richAccordion* aglomerar a varios elementos de interfaz en cada una de sus pestañas. Similarmente al richAccordion, para especificar que una pestaña forma parte de un richTabs, se debe seleccionar de la enumeración *content* que forma parte del *compositeUIElement* el valor *withinARichTab.*

De los metamodelos de contenido y estructura presentados, se derivan los perfiles, que son extensiones al lenguaje UML, para agregar las características propias de MoWebA y por ende hacer posible la representación de la sintaxis concreta de MoWebA que se presenta a continuación en la siguiente sección.

**3.3 - El perfil para el modelado de contenido en MoWebA (Content profile).**

En la Figura 3 se muestra el perfil de contenido para el modelado de los PiM de contenido de la aplicación con MoWebA. Como puede apreciarse, los elementos resaltados en azul forman parte de la extensión que se hace en este trabajo de tesis. Los elementos compuestos del perfil (*compositeUIElements*) son representados por medio de metaclases y estereotipos. Los elementos simples para la representación de contenido son extendidos por medio de propiedades estereotipadas. A continuación se presentan los nuevos elementos simples y compuestos propuestos a MoWebA.

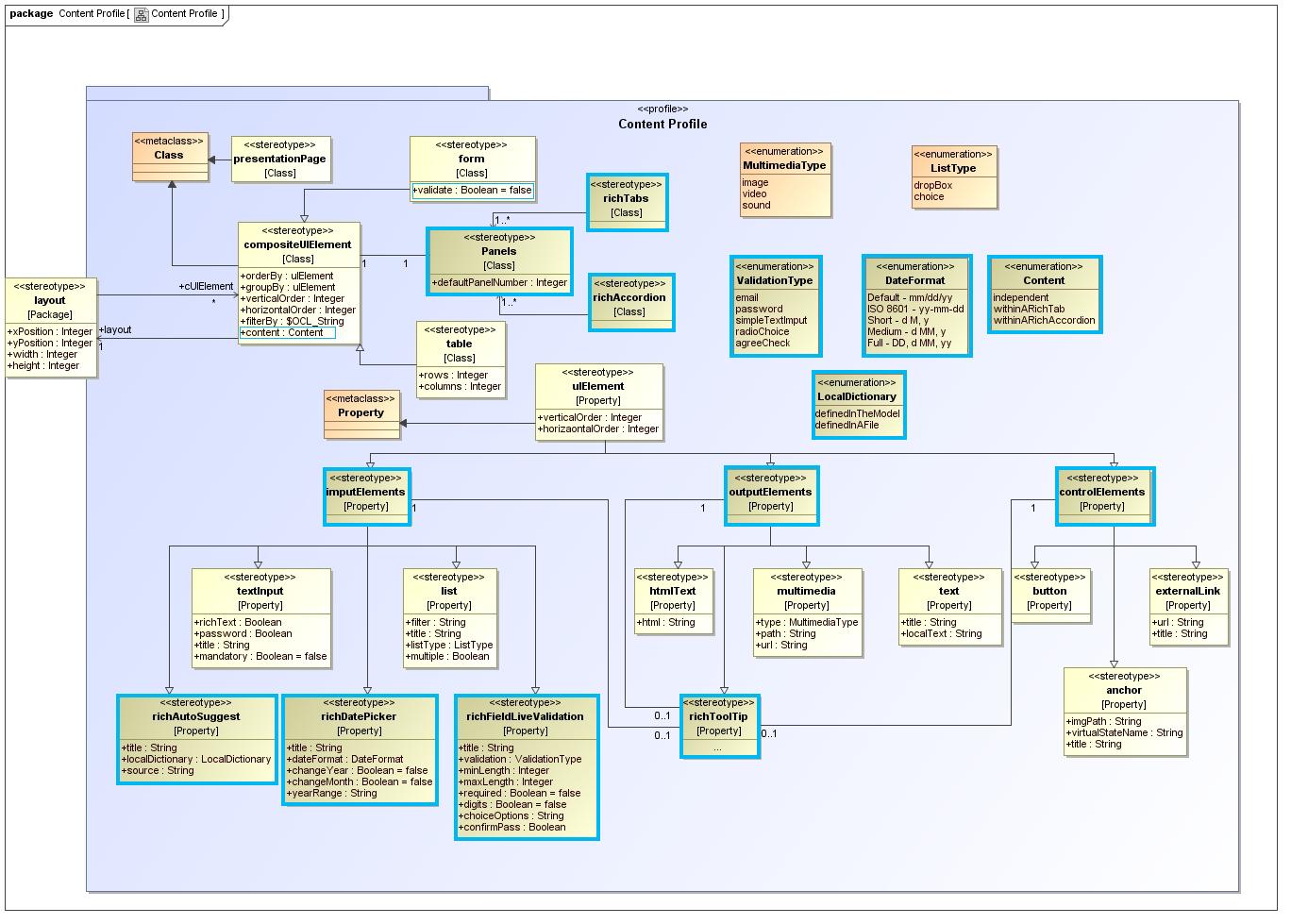
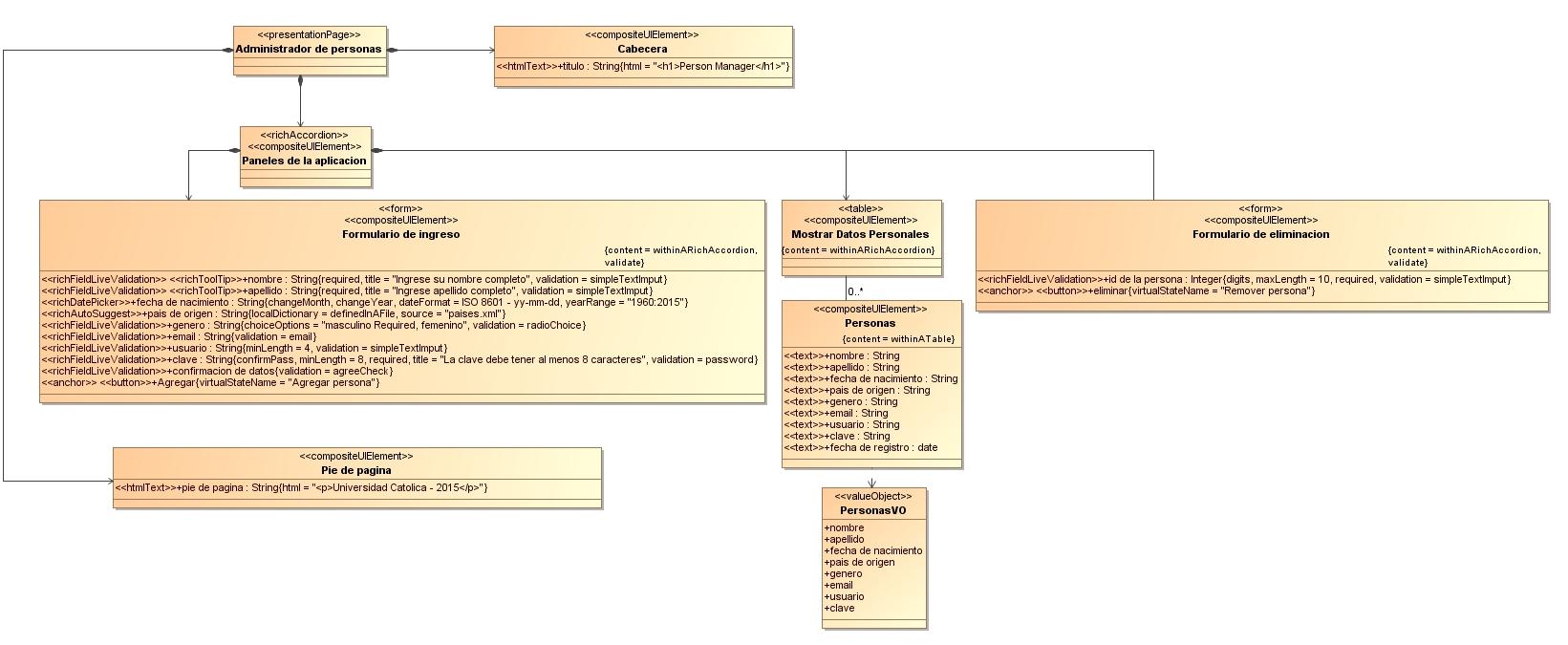
****

Figura 6 Perfil de contenido de MoWebA

**3.4 Modelado de interfaces de usuario con MoWebA.**

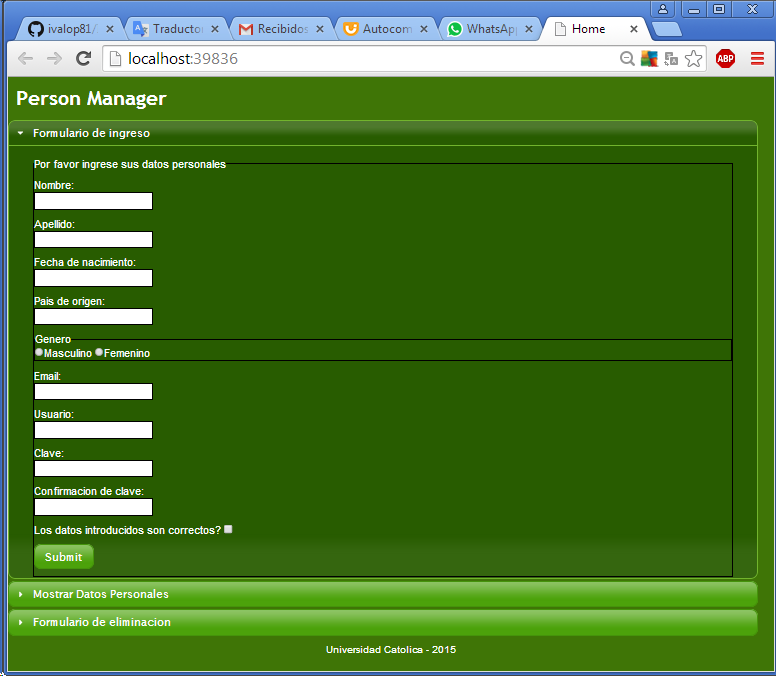
En esta sección se presenta un ejemplo simple de modelado con las extensiones llevadas a cabo a MoWebA, con la idea de dar a conocer la manera en que se implementan los modelos independientes de la plataforma (*PIM*) de la propuesta de este trabajo de fin de carrera. En la Figura 4 se presenta el *PIM* de una aplicación de ejemplo a modo de ilustración. Cada uno de los elementos de interfaz que forman parte del *PIM* son representados por clases y atributos que son etiquetados con un estereotipo en particular. Los nuevos elementos de interfaz que son parte de la extensión son antecedidos con el prefijo *rich*. Primeramente se presenta la página *Administrador de personas* que está compuesta del elemento principal *Paneles de la aplicación* que es un *richAccordion*. El *richAccordion* a la vez está compuesto de tres paneles que son: el *Formulario de ingreso*, *Mostrar datos personales* y *Formulario de eliminación*. Cada uno de estos paneles lleva el valor etiquetado *withinARichAccordion,* expresando de esta forma que cada uno de ellos forma parte de un richAccordion. De manera análoga, si se tratase de un elemento de interfaz del tipo *richTabs,* cada uno de los paneles estaría etiquetado con el valor *withinARichTabs*, como se citó en la sección anterior.

Dentro de cada uno de los paneles se encuentran los distintos elementos de interfaz de usuario como los *richFieldLiveValidation*, *richDatePicker*, *richToolTip* y *richAutoSuggest* con sus respectivos valores etiquetados para representar las propiedades de cada uno de ellos. Por ejemplo, el panel *Formulario de Ingreso*, cuenta con el atributo *clave,* que tiene a los estereotipos *richToolTip* y *richFieldLiveValidation* para indicar que al posicionar el puntero del mouse sobre el texto de entrada del campo clave, el mensaje “*La clave debe tener al menos 8 caracteres”* será desplegado y que también este campo tendrá validaciones de longitud mínima de 8 (*minLength=8*), no puede quedar vacío (*required*) y que se va a crear un campo extra de confirmación de clave (*confirmPass=true*).



**Figura 7** PIM de una aplicación de ejemplo con MoWebA

Finalmente en la Figura 5 y la Figura 6, se presentan la vista general de la interfaz de usuario final y algunos de los elementos enriquecidos que son representados por el PIM del ejemplo anterior respectivamente. Las vistas se obtienen una vez que se genera el código fuente correspondiente de la aplicación, a partir de la ejecución de las reglas de transformación que son definidas dentro de una plantilla. La metodología de transformación será presentada en el siguiente capítulo.



**Figura 8** Interfaz final obtenida a partir del PIM de ejemplo



**Figura 9** Elementos enriquecidos y validaciones de campos que forman parte de la aplicación

En este capítulo se presentó primeramente el proceso de desarrollo de la propuesta, la cual incluye la etapa de modelado de los PIM de presentación de una aplicación. La presentación de una aplicación en MoWebA incluye al contenido, que abarca a los distintos elementos de interfaz RIA o tradicionales, como así también la posición o ubicación de estos elementos dentro de las páginas. Los elementos que forman parte de la extensión llevada a cabo a la metodología web MoWebA, precisamente a nivel de contenido, son los widgets richDatePicker, richAutoSuggest, richToolTip, richTabs y richAccordion. También se adicionó a los elementos ya existentes de la metodología, la validación de los campos dentro de un formulario por medio de la extensión richFieldLiveValidation.

Todos estos nuevos elementos primeramente fueron agregados al metamodelo de contenido para la representación de la sintaxis abstracta de cada uno de ellos. A partir de esta definición, se presentó el perfil de contenido, que extiende a UML permitiendo expresar la sintaxis concreta de MoWebA. En el perfil de contenido, se describió cada uno de los nuevos elementos agregados, con el detalle de cada uno de sus valores etiquetados, que son necesarios para expresar las características que van a tener los widgets, como así también el elemento de validación de campos. También se presentó el perfil de posicionamiento, en donde se mostró como se establecen las coordenadas de cada uno de los elementos.

Finalmente se ilustró un ejemplo de PIM con las extensiones RIA propuestas a MoWebA, junto a algunas vistas de tomas de pantalla de la aplicación.

**Capitulo 4**

**La Transformación de modelo a texto (M2T) para el diagrama de contenido de moweba**

**4.1 Introducción**

En capítulo anterior se ha visto el metamodelo de contenido (*Content*) para la representación de la sintaxis abstracta de MoWebA con la nueva clasificación de elementos de interfaz, en la cual aparecen a diferencia de MoWebA tradicional que contempla elementos para la web 1.0, elementos enriquecidos que son comunes en las aplicaciones RIAS. También junto al metamodelo de contenido, se presentó el metamodelo de estructura (*Layout*), que permite definir la posición de los diferentes elementos del metamodelo de contenido, dentro de una página en particular. Al finalizar la presentación de los metamodelos, se dio pié a los perfiles para el modelado de los *PIM* de la aplicación, extendiendo a UML con las características nuevas propias del DSL MoWebA, específicamente a lo concerniente a contenido y posición.

En este capítulo, en primera instancia se presentarán algunos conceptos básicos acerca de la generación de código y los beneficios que aportan al proceso de desarrollo basado en modelos. El proceso M2T puede tener varios objetivos finales, como la generación, a partir de los modelos de documentación, código fuente, o cualquier componente de software necesario en una aplicación. Aquí nos concentraremos en la generación de código fuente para la interfaz de usuario de una aplicación web representada por medio de modelos que han sido definidos en MoWebA. Una revisión de algunas de principales las herramientas M2T existentes en la actualidad se dará a conocer, para luego presentar a la herramienta M2T Acceleo que es la herramienta utilizada para la generación de código. Estas secciones están basadas en libro de Brambilla y otros[[].

Seguidamente se presentará a las plantillas (*templates*) para llevar a cabo la transformación de los PIM de entrada de MoWebA correspondiente al contenido y a las plantillas de transformación para establecer la configuración de las posiciones de cada uno de los elementos del PIM.

Finalmente se concluirá el capítulo con un ejemplo de transformación M2T para PIM modelados con MoWebA.

En el enfoque tomado en este trabajo, la generación de código es total a partir de los modelos de los PIM de entrada, para los elementos de la capa de presentación de MoWebA.

En el contexto de este trabajo de fin de carrera, a generarse a partir de los modelos será HTML, Javascript para la representación de los distintos elementos de interfaz de MoWebA como así también CSS para el posicionamiento de estos elementos en las páginas.

Para este trabajo de tesis, la herramienta M2T basado en plantillas Acceleo, será la protagonista de llevar a cabo la transformación M2T de los modelos de entrada.

**4.5  LA HERRAMIENTA DE TRANSFORMACIÓN M2T ACCELEO**

*Acceleo* posee varias características que la hacen interesante para la generación de código a partir de los modelos de entrada. *Acceleo* es un generador de códigos de uso abierto (*open source*). Como tal es posible utilizarlo, bifurcarlo y contribuir con la evolución del proyecto. Cuenta con una gran comunidad (*Eclipse Foundation*) que la mantiene.Está integrado con el IDE del *Eclipse*, un editor robusto, con corrector de sintaxis, detección de errores en tiempo real, soluciones rápidas, refactorización y mucho más. También contiene vistas dedicadas que ayudan a navegar amigablemente por el generador de código**.**

Por lo general, con el generador de código, es fácil perderse en el código generado. De manera a manejar este inconveniente, Acceleo contiene un motor de trazabilidad que permite encontrar fácilmente, que elementos del modelo y que parte del generador (plantilla de transformación) han sido utilizados para generar la pieza de código. Generadores de código son a menudo limitados a un conjunto de tecnologías. Con el enfoque basado en *templates*,  *Acceleo* puede generar código para cualquier tipo de lenguaje. Si es posible escribir la plantilla de transformación, *Acceleo* puede generar el código correspondiente.

En algún momento podría considerarse adecuado modificar  manualmente el código generado por el *template* de transformación  y mantener las modificaciones manuales realizadas, en caso que se desea regenerar el código de la aplicación. Acceleo provee de tal flexibilidad, permitiendo llevar a cabo generaciones incrementales.

**4.6 METAMARCADORES DE ACCELEO**

Para obtener los datos de los modelos UML de entrada, Acceleo hace uso de los metamarcadores (*tags*). Dentro de los metamarcadores más importantes pueden citarse los siguientes a continuación:

**Archivos:**Para generar código, los archivos deben ser abiertos, rellenados y posteriormente cerrados. En Acceleo, existe un *tag* de archivo (*file*) dedicado, el cual es utilizado para imprimir contenido creado entre el comienzo y el final del *tag file*. La ruta y el nombre de archivo son definidos por un atributo del *tag*.

**Estructura de control**: Existen *tags* para definir estructuras de control tales como *loops* (*for*) para iterar entre colecciones de elementos y en ramas condicionales (if tag).

**Consultas:** Las consultas OCL pueden ser definidas por medio del *tag query*. Los queries pueden ser llamados a través de todo el *template* y pueden ser utilizados para factorizar código que es recurrente.

**Expresiones:** Existen expresiones generales para incluir los valores de las expresiones computadas en el texto generado, de manera a producir las partes dinámicas del texto de salida. Las expresiones pueden también ser utilizadas para llamar a otros *templates* y de esa forma, incluir el código generado por el *template* llamado en el código producido por *template* llamante. La invocación a otros *templates* puede ser comparado a los métodos en Java.

**Áreas protegidas:** Un concepto especial llamado *protected area* ha probado ser útil y es soportado por el Acceleo utilizando el *tag protected*. Los *protected areas*, se emplean para marcar secciones en el código generado que no deben ser sobrescritos de nuevo, luego de una nueva ejecución del generador de código. Esta sección típicamente contiene código manualmente escrito.

**4.7 Transformacion a codigo de los pim de moweba con Acceleo.**

Acceleo propone un ambiente ameno de trabajo basado en el IDE del Eclipse. Uno puede seleccionar la vista propia del Acceleo en el IDE y obtendrá un ambiente personalizado de trabajo con todas las características anteriormente citadas, en donde se podrá ver el editor de plantillas de transformación, la grilla de propiedades, la grilla de errores y la barra exploradora en donde es posible navegar sobre un proyecto el formato de árbol de expansión. En él se encuentran las plantillas de transformación, los modelos de entrada en formato XMI y los módulos de servicio de Java para complementar a las plantillas de transformación.

Para poder llevar a cabo las transformaciones sobre los modelos de MoWebA, se tuvieron en cuenta las siguientes herramientas para el proceso de desarrollo con el Acceleo:

IDE Eclipse Kepler Service release 2

Acceleo Versión 3.4

UML Designer for Eclipse Kepler version 3.0

**4.7.1 Transformación de los modelos de MoWebA de MOF a EMF UML2 (v2.x) XMI.**

Teniendo en cuenta que Acceleo solamente puede des-serializar modelos de entrada UML en el formato EMF UML 2, es necesario primeramente exportar el proyecto con los modelos PIM y perfiles UML desde la herramienta Magic Draw 16.0 en la cual fueron modelados en primera instancia. Una vez llevado a cabo este paso, el proyecto es importado al Acceleo y de esta forma se tienen los modelos PIM junto a los perfiles UML en la versión UML2 que son los elementos de entrada a la herramienta de transformación, que serán posteriormente des-serializados por medio de las plantillas.

El enfoque tomado para llevar a cabo las transformaciones se basa principalmente en dos plantillas de transformación. La primera de ellas, la plantilla de contenido, se encarga de transformar a los distintos elementos de interfaz que han sido definidos por medio del perfil de contenido de MoWeba. Dependiendo de sí el elemento modelado, es un elemento de interfaz RIA o no, se generará el archivo HTML correspondiente a la página, con la sección *Javascript*, encerrada con las etiquetas *script* o no. Solamente los elementos que forman parte de la extensión propuesta a MoWebA presentan código Javascript para la librería jQueryUI y jQuery Form Validate. Por supuesto, dependiendo del elemento RIA definido, el código Javascript generado, presentará características propias del elemento y comportamientos que fueron definidos en el modelo PIM de contenido.

Por otro lado se encuentra la plantilla de estructura, que transforma las posiciones definidas en pixeles, a cada uno de los *compositeUIElement* en un archivo .css con las coordenadas de posicionamiento correspondiente a cada uno de ellos. A continuación se presentaran las plantillas de contenido y estructura respectivamente.

**4.7.2 Plantilla de transformación para los elementos del perfil de contenido.**

Esta plantilla tiene la responsabilidad de llevar a cabo la transformación de los distintos elementos de interfaz definidos en el perfil de contenido. Dentro de los elementos definidos en el perfil de contenido, tenemos a los elementos que no tienen propiedades enriquecidas y que no tienen características interactivas. Estos elementos son los correspondientes a los de la web 1.0 y son representados por medio de etiquetas y atributos HTML en el cuerpo *(body).*

Por otro lado se encuentran los elementos con propiedades enriquecidas (RIAS) como los *richToolTip*, *richAccrodion*, *richTabs*, *richDatePicker*, *richAutoSuggest* y los *richFieldLiveValidation* que son parte de la extensión llevada a cabo a MoWebA para este trabajo de fin de carrera. Estos elementos a la par de contar con la sección body del HTML para representar el elemento, también cuentan con una sección *Javascript* (encerradas en el *tag script*) para representar la parte dinámica del elemento. La sección correspondiente al tag script contiene el código jQuery correspondiente al elemento definido. Cabe destacar el punto de que el identificador (*id*) de todos los elementos de interfaz, se establecen por medio del nombre del elemento, sin espacios. La identificación de cada uno de los elementos por medio del *id*, resulta importante, principalmente para los elementos de interfaz RIAS, debido a que permiten machear el código *Javascript* generado para *jQuery* en la sección del *tag script* (correspondiente a la parte dinámica) con el código HTML generado en el *tag body* para el elemento (correspondiente a la parte estática).

Primeramente la plantilla inicia verificando la clase principal del PIM de contenido, esta clase es la clase con el estereotipo *PresentationPage*, que indica el nombre que va a tener la página. Por ende, abre un archivo HTML de salida con tal nombre, en donde todos los elementos de interfaz definidos en el resto de las clases del modelo de clases, estarán contenidos dentro de este archivo. El nombre de la página, junto a las dependencias CSS (correspondientes al posicionamiento de los elementos, obtenidos a partir de la plantilla de posicionamiento y los correspondientes a *jQueryUI* y *jQuery form validation*) y *Javascript* (correspondientes a *jQueryUI* y *jQuery form validation*)*,* están definidos dentro de la plantilla, encerradas dentro del *tag head*.

Seguidamente se definen los componentes correspondientes a los tags *script* (en caso de elementos enriquecidos solamente) y *body* respectivamente del archivo abierto HTML. En la Figura 2, se presenta un ejemplo del proceso de transformación para el elemento *richDatePicker*. En primera instancias se muestra el modelo PIM de entrada, seguido de la plantilla de transformación para la sección *Javascript* y por último la plantilla de transformación donde se define al elemento en sí mismo. Como puede apreciarse en el modelo PIM de entrada, el *richDatePicker* (marcado en celeste), puede estar definido junto a varios otros elementos de interfaz, dentro de la clase que la contiene. Cada uno de los elementos es definido por medio de atributos estereotipados y valores etiquetados específicos.

El estereotipo *richDatePicker* indica que el atributo *fecha de nacimiento*, es un calendario y los valores etiquetados del atributo, definen las características del *datePicker*. Dentro de los valores etiquetados definidos para el atributo *fecha de nacimiento*, tenemos a *changeMonth*, *dateFormat* y *yearRange;* que indican respectivamente que una lista de los meses se agregará al *datePicker*, que el formato de fecha con el cual el cuadro de texto será completado, luego de la selección de una fecha dada en el calendario desplegado, será del tipo *ISO 8601 –yy-mm—dd* y que el rango 1960:2015 será desplegado en una lista.

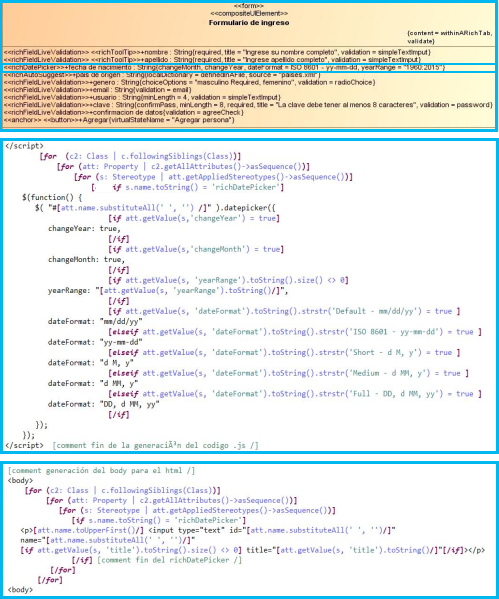


Figura 10 PIM de contenido de MoWebA y templates de trasnformación de contenido

La plantilla de contenido lleva a cabo dos iteraciones completas sobre las clases y sus atributos respectivos definidos en el PIM de presentación. La primera pasada, es para la generación del código *Javascript* correspondiente a los elementos enriquecidos. La otra pasada es para la definición del cuerpo del elemento que corresponde al *tag body*. En cada una de las iteraciones sobre las clases, se verifican sus atributos estereotipados para que en caso de estar definido en el modelo, un elemento en particular, se escriba el código correspondiente al elemento. El nombre del atributo estereotipado de un elemento enriquecido de la clase, se interpreta en la plantilla como el identificador (id) de elemento, y sirve para machear el comportamiento dinámico del elemento con la definición del mismo. Cada uno de los metamarcadores, iteradores y sentencias condicionales, permiten obtener los valores del modelo de entrada, iterar sobre los distintos elementos y preguntar sobre los distintos elementos de interfaz. En la Figura 3 se observa el código HTML generado por la plantilla de transformación para el elemento *richdatePicker* definido en el ejemplo de la Figura 2.

****

Figura 11 Código fuente HTML generado para el *richDatePicker* generado a partir de las plantillas

**4.7.3 Plantilla de transformación para el posicionamiento de los elementos de contenido.**

Con la plantilla de transformación para el posicionamiento de los elementos de interfaz dentro de las páginas, es posible generar el código correspondiente a los *cascading style sheets* (css) a partir de los modelos PIM de estructura (*Layout*). Primeramente al igual que en la plantilla de contenido presentada anteriormente, es necesario importar los servicios Java para poder utilizar dentro de la plantilla, expresiones que no son OCL estándar, como por ejemplo el método *hasStereotype* que permite saber si un elemento UML posee cierto estereotipo para llevar a cabo decisiones. Seguidamente se decide el nombre y la extensión del archivo de salida por medio del *tag file* y dentro de este *tag* comienza el proceso de recorrido dentro los elementos del tipo *package*, en donde se buscan los valores etiquetados del tipo *cUIElement*. Para cada uno de los valores etiquetados *cUIElement* encontrados dentro de unpaquete estereotipado con *Layout*, se agregan los valores correspondientes a las posiciones definidos en el modelo PIM. Las posiciones a definirse corresponden a los valores en pixeles del *height*, *width*, *xPosition(left)* y *yPosition(top)*. En la figura 4 se presenta el template de transformación de estructura.

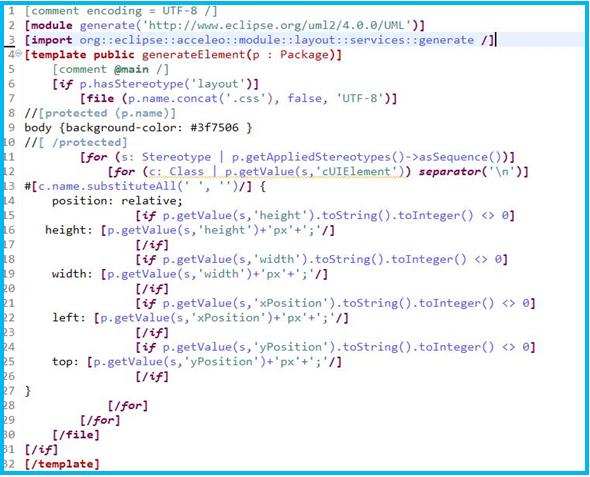


Figura 12 Plantilla de transformación para el posicionamiento de elementos

**4.8 Conclusiones**

En este capítulo se presentó una introducción a los conceptos básicos de las transformaciones M2T. Se dio una breve presentación de los elementos existentes en las plantillas de transformación como así también a algunas de las herramientas existentes en la actualidad para llevar a cabo la transformación de modelo a texto. Entre las herramienta existentes encontramos a XSLT, JET, XPAND, MOFScript y Acceleo. Para este trabajo de fin de carrera, se utilizó a Acceleo para llevar a cabo las transformaciones sobre los modelos UML de entrada, debido a su herramienta de trabajo que actualmente se encuentra bastante evolucionada y debido al soporte que ofrece su comunidad abierta que es parte de la *Eclipse Fundation*. Luego se dio pié a las principales características de Accelo junto a una descripción de los metamarcadores más importantes que pueden utilizarse dentro de las plantillas y que facilitan la des-serialización de los modelos de entrada, que corresponden al código que será generado dinámicamente.

Por último, se describieron a las plantillas de transformación de contenido y posicionamiento de MoWebA, que permiten generar el código (HTML, *Javascript para jQueryUI y jQuery Form Validate)* correspondiente a los distintos elementos que pueden ser definidos en el PIM de presentación, y como estos elementos una vez definidos pueden ser posicionados dentro de las páginas.

**Capítulo 5**

**Ilustración de la propuesta con un enfoque de caso de estudio comparativo**

**5.1 INTRODUCCIÓN**

En este capítulo se describirá la experiencia del autor de la tesis en el proceso llevado a cabo para ilustrar el caso de estudio, en el cual se compara la capa de presentación de la metodología web MoWebA con extensiones RIA, con respecto a la misma metodología sin extensiones. La comparativa entre los enfoques tomados, se enmarca contextualmente en el dominio de las aplicaciones web, precisamente con la obtención de los datos analíticos, en base a la implementación de un sistema de administración de personas o *Person Manager*.

En la actualidad existen diversos métodos empíricos para llevar a cabo validaciones formales sobre algún fenómeno en particular, entre los que se pueden citar a los experimentos, las encuestas y los casos de estudio. Es común en el campo de la ingeniería de software emplear a los casos de estudio como métodos de validación, debido a su flexibilidad y a la posibilidad de tener un mejor control sobre las variables de medición, a costas de un mayor esfuerzo en la interpretación de los resultados obtenidos.

Según Runeson [], un caso de estudio es llevado a cabo para investigar una sola entidad o un fenómeno en su contexto de la vida real, en un espacio de tiempo específico. Típicamente el fenómeno puede ser difícil de distinguir de su ambiente y el investigador debe colectar información detallada sobre un proyecto durante un periodo sostenido de tiempo. Durante la realización del caso de estudio, una variedad de procedimientos de colección de datos y perspectivas de análisis deben aplicarse.

Sin embargo, no siempre es posible tener el rigor suficiente para emplear un caso de estudio y es allí, donde surge como una alternativa; la ilustración, que a pesar de no ser un método de validación formal, sirve para presentar a una audiencia como funciona un artefacto y como este puede resolver un *toy problem* en particular, que permitirá de esta forma, llegar a una conclusión intuitiva [<r2014>]

En este capítulo, se ilustrará el aporte hecho a MoWebA presentando las extensiones realizadas a la metodología, por medio de la resolución de un *toy problem* denominado *Person Manager* desde los enfoques mencionados anteriormente. Para brindar a la ilustración propuesta una mayor formalidad, se seguirán las guías de caso de estudio propuestas por Runeson

**5.2 DISEÑO DE LA ILUSTRACIIÓN DEL CASO DE ESTUDIO**

**5.2.1 Razón fundamental**

Esta ilustración se presenta con la idea de llevar a cabo un análisis crítico de las extensiones RIA llevadas a cabo a la metodología web MoWebA, que forman parte del trabajo de fin de carrera del autor. Esta propuesta de extensión se basa principalmente en proveer a MoWebA de características enriquecidas a nivel de la interfaz de usuario, que le permitirán mantenerse vigente con respecto a las nuevas tendencias de las aplicaciones web de hoy en día, que demandan una mayor interactividad y riqueza en las interfaces de usuario.

Estas características RIA implementadas a MoWebA, ofrecen al usuario final la capacidad de llevar a cabo validaciones locales de datos en un formulario, inserciones de fechas de una manera amigable, resaltar un campo o texto de relevancia con un mensaje personalizado; como así también contraer y expandir información para una mejor administración del espacio en las páginas, que son características comunes en las aplicaciones web RIA. Teniendo en cuenta, las potenciales mejoras que pueden llevarse a cabo con las extensiones propuestas, el propósito de este capítulo es resaltar estas nuevas características por medio de la implementación de un sistema que refleje estas extensiones y a la vez que sirva para poder efectuar un análisis comparativo de los cambios realizados a MoWebA, con respecto a la propuesta de presentación original.

**5.2.2 Objetivos**

La meta principal de la presente ilustración, es obtener datos lo suficientemente reveladores; qué permitan intuir que, la propuesta de extensión a nivel de la capa de presentación para el lado del cliente, llevada a cabo a la metodología web MoWebA, ofrece cobertura a algunas de las diversas características que contemplan las RIA que han sido analizadas en el capítulo 2. Puntualmente, estas características abarcan a la lógica de negocios en el lado del cliente, específicamente a las validaciones locales de campos en un formulario y a las presentaciones enriquecidas que contempla a ciertos eventos en el lado del cliente, *widgets* interactivos y el paradigma de una sola página o single *page paradigm*. El objetivo es analizar estas características, por medio de la resolución de un *toy problem* denominado *Person Manager* , que es una aplicación web que contiene en sus especificaciones funcionales, características de las RIAS y resulta lo suficientemente expresiva para ilustrar la propuesta de extensión.

**52.3 El caso y las unidades de análisis**

El caso ilustrativo está basado en un sistema de administración de personas (*Person Manager*) en el dominio de las aplicaciones web, que fue elegido entre varias otras opciones, debido a que sus requerimientos funcionales ofrecen la posibilidad de representar a todas las características RIA que han sido agregadas a la metodología MoWebA, de una manera clara y sencilla.



Figura 1 Ilustración del sistema *Person Manager* implementado con MoWebA desde dos enfoques distintos

El caso fue analizado desde dos unidades de análisis, en la cual la primera de ellas (el *método A*); consistió en implementar la capa de presentación del *Person Manager*, con MoWebA sin extensiones RIAS y la otra (*el método B*) consistió en la implementación de misma capa de presentación del caso, con la nueva propuesta de extensión RIA a MoWebA. Por lo tanto se trata de una ilustración de caso de estudio embebido, ya que se cuenta con más de una unidad de análisis para un mismo caso, como se puede apreciar en la Figura 1.

A continuación se presenta la descripción del sistema *Person Manager* de una manera general, esto es; las funcionalidades básicas que deben contemplar ambas unidades de análisis. Seguidamente, se agregan al *Person Manager* algunos requerimientos adicionales, que son precisamente, requerimientos RIAS. Las unidades de análisis representan a los dos métodos implementados con MoWebA (A y B) que servirán para obtener respuestas a las preguntas de investigación que se presentaran en la siguiente sección. El *Person Manager* está basado en el trabajo de Gharavi [<sv2008>]

*Sistema de administración de personas (Person Manager)*

*Person Manager es una aplicación, que contiene funciones de creación, listado y borrado de registros correspondiente a personas. La aplicación cuenta con las siguientes vistas:*

***Agregar Persona (Add):*** *es una vista utilizada para capturar la suficiente información acerca de una persona para posteriormente agregarla a una base de datos. En la vista, la información detallada de una persona es ingresada por medio de un formulario. Al presionar el botón enviar (submit button), los datos ingresados se insertan en una base de datos. La vista Agregar Persona, cuenta con los siguientes campos de entrada de texto; nombre, apellido, fecha de nacimiento, país de origen, email, usuario, clave, confirmación de clave; también contiene al campo de selección genero (que puede ser masculino o femenino) y una caja de verificación de datos correctos (checkbox).*

***Listar Personas (List):*** *Consiste en una vista en la cual es posible despliegar todos los datos correspondientes a las personas existentes en la base de datos en una tabla. La tabla contiene una columna por cada campo de información que ha sido completado por un usuario en la vista Agregar Persona .*

***Borrar Personas (Remove):*** *se trata de una vista para borrar a una persona de la base de datos. En un formulario, el id de la persono a borrar es ingresado en un cuadro de texto .Al presionar el botón eliminar, el registro de la persona con el id especificado es eliminada de la base de datos y por ende desaparece de la vista Listar* .

Adicionalmente a la definición común del Person Manager para ambas unidades de análisis mencionadas anteriormente, se adicionan los siguientes requerimientos RIAS, a ser tenidos en cuenta por el enfoque B. Las características RIAS deseables son las siguientes:

*Para los campos de la vista Agregar se requiere:*

* *Para el campo fecha de nacimiento de la persona, se desea que el ingreso de la fecha sea ágil e interactiva y que no sea necesario escribirla manualmente.*
* *En ciertos campos de entrada de texto, se requiere información adicional interactiva que debe desplegarse al posicionarse el ratón sobre el campo. Esto es necesario llevar a cabo en los siguientes campos:*
  + *Nombre: al posicionar el puntero del mouse sobre el campo nombre se debe desplegar el mensaje complementario “Ingrese su nombre completo”*
  + *Apellido: al posicionar el puntero del mouse sobre el campo apellido se debe desplegar el mensaje complementario “Ingrese apellido completo”*
  + *Clave: al posicionar el puntero del mouse sobre el campo clave se debe desplegar el mensaje complementario “La clave debe tener al menos 8 caracteres”*
* *En el campo de entrada de texto País de origen, a medida que caracteres son introducidos, sugerencias deben desplegarse en base al patrón actual, permitiendo al usuario, navegar en tales sugerencias, hasta elegir la opción deseada.*
* *Las siguientes validaciones locales (lado del cliente) de campos del formulario pueden llevarse a cabo:*
  + *Validaciones de campos mandatorios: Para que el formulario pueda ser validado, los campos de entrada de texto nombre, apellido, clave, y confirmar clave, deben ser completados de manera obligatoria, también, para el radio choice género, debe seleccionarse una opción de las existentes (masculino o femenino) . Finalmente, el checkbox de confirmación de acuerdo sobre los datos introducidos debe marcarse..*
  + *Longitud mínima de caracteres en los siguientes campo de entrada:* 
    - *Usuario: la longitud mínima debe ser de dos caracteres.*
    - *Contraseña: la longitud mínima debe ser de 8 caracteres.*
  + *Validación de un email en el formato correcto para el campo Email.*
  + *Deben coincidir los dos campos de entrada del tipo clave, clave y la confirmar clave.*

*Para el campo de la vista Eliminar se requiere:*

* *Para el campo id, se permite solamente el ingreso de números naturales (dígitos) (0,1,….,9, ..).*
* *Longitud máxima de caracteres en el campo de entrada id de la persona es de 10.*
* *El campo Id, es mandatorio.*

*La interfaz de usuario en forma general, debe ofrecer un aspecto single page, por lo tanto la navegación por las distintas vistas de la aplicación, no debe implicar un refrescado total de página. Todas las validaciones sobre los campos de entrada de los formularios que forman parte del Person Manager deben llevarse a cabo de manera local(en el lado del cliente). En caso de errores cometidos por no completar un campo (campo mandatorio), el/los mensaje/s de error, serán desplegados al lado del campo faltante, una vez presionado el botón submit del formulario en cuestión. En caso de cometer algún error en el ingreso de los datos, un mensaje de error en línea (al lado del campo en cuestión) se desplegará al desenfocar el campo, indicando al usuario el motivo del error, sin la necesidad de presionar el botón submit.*

**5.2.4 Preguntas de investigación**

De los objetivos anteriormente citados, surgieron las siguientes preguntas de investigación para esta ilustración:

PI1: ¿Consume una mayor cantidad de tiempo modelar la aplicación utilizando el método B que el método A?

PI2: Para la implementación del método B. ¿Es necesaria una mayor cantidad de generaciones de código para la obtención de la interfaz de usuario final, con respecto al método A?

PI3: ¿Qué ventajas aportan las características RIAS presentes en la aplicación implementada con el método B con respecto al método A, desde el punto de vista de las presentaciones enriquecidas?

PI4: ¿Qué ventajas aportan las características RIAS presentes en la aplicación implementada con el método B con respecto al método A, desde el punto de vista de la lógica de negocios en el lado del cliente?

PI5: Para cada una de las vistas del *Person Manager* ¿Que porciones de la interfaz de usuario han sido posibles de generar de manera automática a partir de los modelos, en los método A y B ?

PI6: ¿Que tan independiente de la plataforma destino son los PIM presentados en el método B con respecto al método A?

**5.2.5 Conceptos**

Las variables de medición necesarias para responder las primeras dos preguntas de investigación se definen a continuación:

*TMA*: El tiempo total en minutos, empleado para modelar la interfaz de usuario del *Person Manager* aplicando el método A. Corresponde a la suma de todos los tiempos de modelado de cada una de las vistas implementadas con el método A

*TMB*: El tiempo total en minutos, empleado para modelar la interfaz de usuario del *Person Manager* aplicando el método B. Corresponde a la suma de todos los tiempos de modelado de cada una de las vistas implementadas con el método B

NGA: El número de veces que el código fuente de la aplicación *Person Manager* fue generado hasta obtener la interfaz de usuario final para el método A.

NGB: El número de veces que el código fuente de la aplicación *Person Manager* fue generado hasta obtener la interfaz de usuario final para el método B.

Una generación de código (equivalente a compilar la aplicación), es llevada a cabo para obtener la interfaz final a partir de los modelos de entrada. En cada una de ellas, el código fuente se va refinando a partir de la modificación de los modelos de entrada.

**5.2.6 Métodos de colección de datos**

Este caso de estudio con enfoque ilustrativo, se implementó con una población de un individuo (el autor de la tesis) tanto para la implementación de las diferentes unidades de análisis, como en la colección de los datos), por lo tanto no se llevaron a cabo entrevistas ni encuestas para la colección de los datos de análisis.

Teniendo en cuenta este hecho particular, los datos necesarios para calcular las variables de respuesta, fueron colectados a medida que las unidades funcionales de la aplicación (las vistas) del *Person Manager* eran implementadas. Primeramente se colectó toda la información correspondiente al método A y luego se procedió a la colección de los datos del método B. Para cada uno de los métodos, primeramente se colectaron todos los datos correspondientes a una vista en particular hasta la conclusión de esta. Seguidamente se pasaba a la siguiente vista y se recaban los datos correspondientes y así sucesivamente.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nombre de las vistas** | **TMA** | **NGA** | **TMB** | **NGB** |
| **Agregar persona** |  |  |  |  |
| **Mostrar persona** |  |  |  |  |
| **Remover persona** |  |  |  |  |
| **Totales** |  |  |  |  |

Una vez finalizada la implementación de ambas unidades de análisis del caso, de manera adicional se llevó a cabo un análisis de líneas de código, para medir el tamaño de los proyectos.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Generadas** | **Correctivos** | **Perfectivos** | **Adaptativos** | **Total LDC** |
| **Manualmente**  **Automáticamente** |  |  |  |  |
| **Total LDC** |  |  |  |  |

**5.2.7 Métodos de análisis de los datos**

En vista que el método de comparación es entre proyectos (*croos-proyect*) o proyecto hermano (*sister project*), solo podemos tener una variable para la productividad o calidad por tratamiento. En este caso no se pueden utilizar métodos estadísticos para asegurar si las diferencias entre las variables de respuesta son significativas. Por lo tanto, solo se pueden comparar los valores obtenidos del proyecto de control informalmente con los valores obtenidos del tratamiento hecho al proyecto, según las recomendaciones hechas en Klitchenan.

**5.2.8 Selección del caso y minimización de los factores de confusión**

Como se mencionó anteriormente, se llevó a cabo una ilustración de caso de estudio comparativo, en la cual se optó por un proyecto piloto en el contexto de las aplicaciones web , la cual fue implementada por un mismo equipo de trabajo (el autor). Se optó por el proyecto piloto *Person Manager* la cual es posible representar las nuevas características extendidas a MoWebA de manera clara y concisa y teniendo en cuenta que dentro de los métodos de comparación existentes en la actualidad, según Klitchenan, el cross-project o single-project, es el más incurre en costos( en el caso de esta propuesta el mayor costo es el tiempo de desarrollo), ya que un mismo proyecto debe llevarse a cabo por un mismo equipo (el autor en este caso) con más de un enfoque o unidad de análisis (en esta ilustración, se utilizaron dos unidades de análisis) . Con respecto a los factores tenidos en cuenta para minimizar los factores de confusión, se ha tenido en consideración los siguientes puntos:

1. Se ha llevado a cabo pruebas de modelado y generación de código para cada uno de los elementos que forman parte del perfil de contenido de MoWebA para lograr la mayor familiaridad posible con cada uno de los elementos de interfaz y sus propiedades intrínsecas. Esto fue necesario (a pesar de que el autor haya sido el implementador de las extensiones RIAS en los perfiles de modelado y las plantillas de transformación) para minimizar la cantidad de errores de modelado en cada método implementado y de paso testear la plantilla de transformación y reducir los riesgos de posibles *bugs* que podrían darse en el transcurso de las pruebas llevadas a cabo en el proceso de desarrollo de cada método.
2. Primeramente se empleó completamente el método A hasta obtener la interfaz final del *Person Manager* sin extensiones RIA y luego el método B con extensiones RIA de igual manera. Se tomó esta decisión, ya que si se implementa una vista con el método A y luego la misma vista con el método B, se puede obtener demasiado familiaridad con el modelado de la vista en cuestión, y por ende no sería tan realista la medición de los tiempos de modelado en el método B, ya que estos podrían reducirse.
3. Los métodos A y B fueron utilizados para construir el *Person Manager* con una semana diferencia, implementándolas hasta obtener la interfaz final desde cero.
4. Cuando un método estaba en proceso, se iba implementando cada una de las vistas y a la par se iba recabando los datos. Se pasaba a implementar la vista siguiente de la aplicación, una vez culminada en su totalidad la vista actual con todos los datos analíticos recabados.

**5.2.10 Selección de los datos**

Los datos analíticos fueron obtenidos a medida que el desarrollo que los proyectos a ser comparados se iban desarrollando, para posteriormente, una vez seleccionados de las planillas correspondientes, se pueda proceder al análisis y de esa forma concluir los resultados obtenidos.

**5.2.11 Definición y almacenamiento de los Datos**

Todos los datos recabados en el desarrollo de cada uno de los métodos, fueron almacenados y mantenidos en las plantillas creadas para el efecto..

**5.2.13 Mantenimiento del protocolo de caso de estudio.**

Este caso de estudio no se rige por un protocolo formal de evaluación.

**5.2.14 Reporte y difusión de los datos obtenidos en la ilustración del caso de estudio.**

En esta sección se responderán a las preguntas de investigación presentadas en la sección 5.2.4 que se citan a continuación:

PI1: ¿Consume una mayor cantidad de tiempo modelar la aplicación utilizando el método B que el método A?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nombre de las vistas** | **TMA** | **NGA** | **TMB** | **NGB** |
| **Agregar persona** | 50 | 3 | 56 | 4 |
| **Mostrar persona** | 27 | 1 | 28 | 1 |
| **Remover persona** | 29 | 2 | 30 | 3 |
| **Totales** | 106 | 6 | 116 | 8 |

En la Tabla 1 puede apreciarse que los tiempos de modelado obtenidos en ambas metodología

Tabla 1, Tiempos de modelado y numero de generaciones para cada una de las Vistas del *Person Manager*

PI2: Para la implementación del método B. ¿Es necesaria una mayor cantidad de generaciones de código para la obtención de la interfaz de usuario final, con respecto al método A?

Se pudo apreciar que el método B, deparó en una mayor cantidad de generaciones de código, para obtener la interfaz RIA final. Esta diferencia representa un aumento del 20% de con respecto a la implementación llevada a cabo con el método A. Analizando las generaciones de cada una de las vistas del *Person Manager* de la Tabla 2, se pudo notar que la vista que tuvo que generarse una mayor cantidad de veces fue la vista Agregar Persona, y este dato resulta concordante con lo que puede intuirse preliminarmente, ya que esta vista es la que contiene la mayor cantidad de requerimientos de interfazy por ende existe una mayor probabilidad de cometer fallos en el modelado, lo que incurre en una mayor cantidad de veces que la aplicación debe generarse hasta su depuración final. Para la vista Remover Persona se incurrió en un número mínimamente superior de generaciones de código implementando el método B, precisamente un 10% más que con el método A, pudiendo deberse también, a que la vista con el método B contiene requerimientos de interfaz RIA a diferencia del método A. También el número de generaciones disminuyó en ambos en ambos métodos con respecto a la vista Agregar Persona. En la vista Listar Personas, se tuvo la mínima cantidad de generaciones de código en ambos métodos aplicados, debido a que gran parte de ella es implementada de manera manual.

De los resultados presentados puede intuirse que a mayor requerimientos de interfaz, se requiere una mayor cantidad de generaciones de código para ambas metodologías aplicadas para ir depurando la aplicación, con un leve incremento en el caso de la metodología B y esto podría deberse a que mayor cantidad de detalles a especificar en los modelos de entrada de la aplicación, existe una mayor posibilidad de cometer fallos.

**PI3: ¿Qué ventajas aportan las características RIAS presentes en la aplicación implementada con el método B con respecto al método A, desde el punto de vista de las presentaciones enriquecidas?**

Son diversos los aportes ventajosos con respecto al método A, que pueden distinguirse en la interfaz RIA obtenida por medio del método B. A continuación se presentan algunas ventajas.

***Apariencia de una aplicación single page***

Cada una de las páginas que forman parte de la aplicación *Person Manager* implementada con el método B, son equivalentes a las pestañas pertenecientes a un *widget richTabs*. Por lo tanto cuando se navega en la aplicación, se tiene la sensación de que trata de una aplicación de escritorio, ya que se puede recorrer cada una de las pestañas sin necesidad de un refrescado de página, teniendo toda la información de manera local en una sola página. Esta característica mejora la interactividad con el usuario de la aplicación y el look and feel del mismo. En la implementación llevada a cabo con el método A, cada una de las páginas de la aplicación, está representada por un enlace, y por ende, cada vez que se visita una página de la aplicación, un refrescado total de página se lleva a cabo, perdiéndose de esta forma el concepto de *single page*.

Estas mismas ventajas de manera similar al RichTab, también pueden ser aprovechadas al utilizar la extensión *RichAccordion*

***Widgets interactivos en la interfaz de usuario***

***Datepicker***

El *datepicker* permite desplegar de una manera ágil e interactiva un calendario debajo de la entrada textual que corresponde al campo fecha de nacimiento. Este calendario interactivo ofrece la posibilidad de navegar por los distintos meses del año actual, con las flechas indicadoras izquierda y derecha, como así también, permite seleccionar un mes en particular desplegando una lista de meses. Con respecto a los años del calendario, es posible definir un rango de años que podrán seleccionarse de igual manera de una lista desplegable.

Este *widget* resulta de gran ayuda a los usuarios finales, ya que grafica e intuitivamente permite seleccionar una fecha, evitando a estos, cometer errores innecesarios al digitar una fecha en un formato dado y optimizando su tiempo de interacción con las páginas web

***AutoSuggest***

El *autocomSuggest* ofrece la posibilidad de desplegar un listado de opciones que facilitan al usuario la escritura de texto en un cuadro de texto de entrada. En el campo país de origen, a medida que el usuario va introduciendo caracteres correspondientes al país deseado, interactivamente se despliegan todos los países que coinciden con el patrón introducido, permitiendo navegar de arriba a abajo por medio de un cursor sobre los distintos países. El cursor se resalta con un color diferente a medida que se va recorriendo por los países sugeridos. Una vez que el usuario encuentra el país de origen deseado, al presionar la tecla entrar o al hacer clic sobre el país, este se escribe en el cuadro de texto de entrada.

***ToolTip***

A menudo es útil, complementar con información adicional a las páginas. Con el toolTip un mensaje informativo útil al usuario es desplegado al posar el puntero del mouse sobre un cuadro de texto de entrada en particular. Para el *Person Manager*, en los campos nombre y apellido se muestra un mensaje en el que se indica al usuario que se ingrese el nombre y el apellido completo. Para el caso del campo contraseña, se despliega al usuario a modo de sugerencia, el mensaje de seguridad que indica el ingreso de caracteres alfanuméricos con mayúsculas y minúsculas combinados con caracteres especiales y que contenga por lo menos una longitud de ocho caracteres.

***Optimización del espacio y navegabilidad de las páginas***

Con la extensión *RichTab,* es posible encapsular distintos elementos de interfaz presentes en MoWebA, tales como entradas de texto, enlaces, botones, textos, hipervínculos, formularios y tablas, como así también elementos enriquecidos que son parte de la extensión RIAS, propuesta a MoWebA; como los *RichDatePicker*, *RichToolTips*, *RichAutocompletes*. La extensión de validación de campos *RichFieldLiveValidation*, también puede ser utilizada dentro de un RichTab, para la validación de los campos en un formulario. La posibilidad de encapsular muchos elementos de interfaz dentro de cada una de las pestañas que forman parte de un *RichTab*, resulta ventajoso con respecto a la optimización espacial de los elementos dentro de las páginas.

Estas mismas ventajas de manera similar al RichTab, también pueden ser aprovechadas al utilizar la extensión *RichAccordion*

***PI4: ¿Qué ventajas aportan las características RIAS presentes en la aplicación implementada con el método B con respecto al método A, desde el punto de vista de la lógica de negocios en el lado del cliente?***

Cuando se habla de lógica de negocios en el lado del cliente, hablamos de operaciones complejas y específicas para un dominio en particular, como así también de validaciones sobre los datos de entrada. Las extensiones RIAS propuestas a MoWebA, abarcan específicamente a las validaciones sobre los campos de entrada en los formularios.

***Validaciones locales de los diversos campos de un formulario.***

La ventaja principal de llevar a cabo validaciones en los formularios de manera local, es que no es necesario ninguna interacción con el lado servidor, lo cual mejora el rendimiento de la aplicación, evitando retardos al recargar la página tras la solicitud de envío de los datos. Con el elemento *RichFieldLiveValidation* es posible llevar a cabo validaciones a los diversos campos de los formularios de la aplicación *PersonManager*. Dentro de las validaciones que se han efectuado se muestra primeramente la validación en los campos que son mandatorios y que no pueden quedar vacios.

Seguidamente se efectuaron controles locales sobre la cantidad de caracteres que deben tener como mínimo algunos campos, tales como los de usuario, clave y confirmación de clave, que se han configurado como mínimo en 2 y en 8 caracteres respectivamente. En contraparte para los campos nombre y apellido se verificó que estos no excedan una cantidad máxima de 30 caracteres. Para los campos clave y confirmación de clave, también se verificó que ambos coincidan en los valores introducidos.

Para los campos numéricos, se valida que solamente sea posible el ingreso de dígitos (valores del 0 al 9), por ejemplo; en el campo id, utilizado para borrar un registro del sistema. En este campo de igual manera no es posible ingresar más de 12 dígitos para evitar algún desbordamiento numérico en la base de datos. También, el campo email verifica que la cadena ingresada por el usuario corresponda a un email válido.

Por último, en el campo de selección género, es mandatorio seleccionar uno de los radio controles (masculino, femenino), como así también, es mandatorio seleccionar la caja de selección del campo de conformidad. Los datos introducidos en el formulario solo serán enviados al servidor, cuando todos los campos pasen la validación correspondiente a cada uno de ellos.

**5.3 ASUNTOS LEGALES, ÉTICOS Y PROFESIONALES**

Puesto que la ilustración de caso de estudio es parte de un proyecto de tesis, todos los datos obtenidos quedan regidos bajo las normas expuestas por la universidad. Teniendo en cuenta que una sola persona implementó las unidades de análisis y a la vez recabó los datos analíticos, se trató de llevar adelante cada paso, con la mayor transparencia y objetividad posible, para que los resultados obtenidos sean fidedignos y de valor. Sin embargo esto no es suficiente para otorgar la suficiente formalidad a los resultados obtenidos. Es bajo esta circunstancia, que decidió llevar a cabo una ilustración y no un caso de estudio, ya que los resultados y conclusiones obtenidas, se dejan a la intuición y criterio de la audiencia, y no cuentan con el rigor que conlleva un caso de estudio.

**5.4 CONCLUSION**

En esta sección, se presentó una ilustración con un enfoque de caso de estudio comparativo ente proyectos. Para tal motivo se implementó una aplicación denominada *Person Manager* en la cual puede reflejarse muchas de las extensiones RIA llevadas a cabo a la metodología web MoWebA, que es el aporte principal de este trabajo de fin de carrera. El Person Manager fue implementado desde dos enfoques distintos, precisamente para modelar la presentación que abarca al contenido y estructura de las páginas. En uno de ellos se utilizó a la metodología MoWebA en su forma original. En el otro enfoque se utilizó a MoWebAcon las extensiones RIA.

El *Person Manager* fue separado en tres vistas que fueron analizadas y en la cual se recabaron datos analíticos para responder a preguntas de investigación. Esos datos correspondían

**CAPITULO 6**

**CONCLUSIONES Y TRABAJOS FUTUROS**

En este trabajo de fin de de carrera se llevó a cabo una investigación acerca de las RIAs, analizando sus características y las tecnologías utilizadas para su implementación. Se ha visto que dentro de las características más relevantes, se encuentra el uso del lado del cliente para mejorar la interactividad de las aplicaciones para con los usuarios y de esa forma, obtener aplicaciones web similares a una aplicación de escritorio, con pocos retardos en las recargas de página. Dentro de las tecnologías RIAs existentes, las basadas en librerías *Javascript* son las más populares y la más utilizada actualmente es *jQuery*. Con las extensiones a *jQuery*, *jQueryUI* y *JQuery validation plug-in* es posible explotar el lado del cliente agregando *widgets* interactivos como los richAccordion, richTabs, richAutoSuggest, richToolTip, richDatePicker y llevar a cabo validaciones locales de campos con el *richFieldLiveValidation*.

Hoy en día, la ingeniería de software basada en modelos, cumple un papel importante en el desarrollo del software. He allí que existen muchas metodologías web con este enfoque y que dan cierta cobertura a características de las RIAs. En base a una investigación del estado del arte de las principales metodologías web basadas en modelos, se concluyó que ninguna de ellas logra dar cobertura a todas las características RIA. He allí la necesidad de extender a las metodologías web existentes con nuevas características RIAs o bien proponer nuevas metodologías que ofrezcan una mayor flexibilidad de extensión. En este trabajo se propuso a la aproximación MoWebA que está basada en los estándares propuestos por la OMG, como la alternativa apropiada, debido a su adecuada separación en capas y baja cohesión, que le brinda la posibilidad de llevar a cabo extensiones RIAs a cualquiera de las capas sin afectar a las otras. En este trabajo se extendió con características RIAs a la capa de Presentación de MoWebA que abarca a los metamodelos de Contenido y Estructura.

Precisamente con este trabajo de fin de carrera se alcanzó aportar lo siguiente:

* Un estudio detallado de las principales características y tecnologías de las RIAs
* Una investigación del estado del arte de las principales metodologías web basadas en MDD y MDA que ofrecen cobertura a las RIAs.
* Un análisis de los elementos de interfaz enriquecidos (*widgets*) más utilizados.
* Una extensión al metamodelo de contenido de MoWebA agregando los siguiente:
  + Una reestructuración y clasificación de los elementos de interfaz con los que cuenta la aproximación MoWebA, separando a los elementos en elementos de entrada, salida y control.
  + *Widgets interactivos* comunes en las aplicaciones RIAs, precisamente richAccordion, richTabs, richAutoSuggest, richDatePicker y richToolTip y el richFieldLiveValidation.
* Para la definición de la sintaxis concreta de la presentación, se agregaron los nuevos widgets al perfil de Contenido de MoWebA.
* Un análisis de las principales herramientas de transformación de modelo a texto (M2T) basado en plantillas.
* Con la herramienta de transformación (M2T) Acceleo se implementaron las siguientes plantillas:
  + **La plantilla de presentación**, la cual genera código para cada uno de los elementos definidos en el perfil de contenido de MoWebA a partir de los PIM de entrada. Para los *widgets* se genera código para la plataforma jQueryUI y jQuery validation plug-in.
  + **La plantilla de estructura**, la cual genera código CSS con las posiciones establecidas en el PIM de entrada
* Una ilustración de caso de estudio en la cual se validan los aportes realizados a la capa de presentación de MoWebA.

Finalmente, en la tabla se presentan las características RIAs y a las metodologías web contempladas en el estados del arte junto a la aproximación MoWeba luego de las extensiones RIAs.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Características versus metodologías | | OOHDM-RIA | OOH4RIA | WebML - RIA | Patrones con UWE | Patrones OOWS | UsiXML | UWE-R | Espacios interactivos con UML | UWE + RUX | MoWebA |
| 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 | 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 | si |
| Paradigma de página única | si | si | si | si | - | si | - | - | si | si |
| Contenido multimedia | - | si | - | - | - | 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 | - | - | - |

Como trabajos futuros se podrían llevar a cabo lo siguientes extensiones RIAs a MoWebA:

* Agregar más propiedades a los *widgets* que forman parte de la extensión al metamodelo de contenido (*richDatePicker*, *richAutoSuggest, richAccordion, richTabs, richToolTip y richFieldLiveValidation)* que ofrecen las plataformas jQueryUI*[[39]](#footnote-39)* y jQuery Validation Plugin[[40]](#footnote-40).
* Agregar más *widgets* interactivos al metamodelo de contenido como por ejemplo el *dialog*, *menú,* *progressbar*, *selectmenú*, *slider* y *spinner*.
* Agregar a la capa de navegación extensiones para ofrecer una interacción asíncrona entre los lados cliente y servidor para cubrir el refrescado parcial de las páginas.
* Separar en el modelo de dominio las entidades, que pueden ser alojadas en lado cliente y servidor para lograr persistencia de datos en el lado cliente.
* Efectuar la validación de la propuesta de extensión RIA a la capa de presentación de MoWebA, con un caso de estudio formal, detalladamente planead, en la que participe una población más grande de personas. Esto sería interesante ya que se minimizarían en gran medida las amenazas a la validez existentes en la ilustración de caso de estudio actual.

1. La evolución de la web: [http://www.evolutionoftheweb.com/?hl=es](http://www.evolutionoftheweb.com/?hl=es%20) [↑](#footnote-ref-1)
2. Google: <http://www.google.com> [↑](#footnote-ref-2)
3. Wikipedia: <http://www.wikipedia.org> [↑](#footnote-ref-3)
4. Facebook: <http://www.facebook.com> [↑](#footnote-ref-4)
5. Twitter: <https://twitter.com/> [↑](#footnote-ref-5)
6. Youtube: http://www.youtube.com/ [↑](#footnote-ref-6)
7. Object Management Group: <http://www.omg.org/> [↑](#footnote-ref-7)
8. Departamento de Ingeniería Electrónica e Informática de la Universidad Católica Nuestra Señora de la Asunción [↑](#footnote-ref-8)
9. Model Driven Architecture: <http://www.omg.org/mda/> [↑](#footnote-ref-9)
10. **Wikipedia- *Rich internet applications*:** <http://en.wikipedia.org/wiki/Rich_Internet_application> 2015 [↑](#footnote-ref-10)
11. **Oracle:** <http://www.oracle.com/technetwork/java/javase/javawebstart/index.html> 2015 [↑](#footnote-ref-11)
12. **Adobe AIR:** <http://www.adobe.com/products/air.html> 2015 [↑](#footnote-ref-12)
13. Describe un estilo de comunicación sobre Internet donde la petición de una transacción se origina en el servidor. [↑](#footnote-ref-13)
14. **Designing Web Interfaces:** <http://designingwebinterfaces.com/essential_controls> 2009 [↑](#footnote-ref-14)
15. **UX BOOTH:** <http://www.uxbooth.com/articles/essential-controls-for-web-applications/> 2010 [↑](#footnote-ref-15)
16. **jQuery user interface:** <http://jqueryui.com/> 2015 [↑](#footnote-ref-16)
17. **jQuery Validation Plugin:** <http://jqueryvalidation.org/> 2015 [↑](#footnote-ref-17)
18. **jQuery:** <http://jquery.com/> 2015 [↑](#footnote-ref-18)
19. **Usage of JavaScript libraries for websites** <http://w3techs.com/technologies/overview/javascript_library/all> 2015 [↑](#footnote-ref-19)
20. **List of Ajax frameworks:** <http://en.wikipedia.org/wiki/List_of_Ajax_frameworks> 2015 [↑](#footnote-ref-20)
21. **MDA** [www.omg.org/mda/](http://www.omg.org/mda/) 2015 [↑](#footnote-ref-21)
22. **UML:** [www.omg.org/spec/UML](http://www.omg.org/spec/UML) 2015 [↑](#footnote-ref-22)
23. **OCL:** [www.omg.org/spec/OCL](http://www.omg.org/spec/OCL) 2015 [↑](#footnote-ref-23)
24. **MOF:** [www.omg.org/mof/](http://www.omg.org/mof/) 2015 [↑](#footnote-ref-24)
25. **QVT:** [www.omg.org/spec/QVT/1.1/](http://www.omg.org/spec/QVT/1.1/) 2015 [↑](#footnote-ref-25)
26. **XSLT:** [www.w3.org/TR/xslt20/](http://www.w3.org/TR/xslt20/) 2015 [↑](#footnote-ref-26)
27. **JET:** <https://projects.eclipse.org/projects/modeling.m2t.jet> 2015 [↑](#footnote-ref-27)
28. **XPAND:** <https://eclipse.org/modeling/m2t/?project=xpand> 2015 [↑](#footnote-ref-28)
29. **MOFScript:** <https://eclipse.org/gmt/mofscript/> 2015 [↑](#footnote-ref-29)
30. **Acceleo:** [www.acceleo.org/](http://www.acceleo.org/) 2015 [↑](#footnote-ref-30)
31. **Google Web Toolkit:** <http://www.gwtproject.org/> 2015 [↑](#footnote-ref-31)
32. **OpenLaszlo:** <http://www.openlaszlo.org/> 2015 [↑](#footnote-ref-32)
33. **Interaction Design Foundation:** <http://www.interaction-design.org/encyclopedia/human_computer_interaction_hci.html> 2015 [↑](#footnote-ref-33)
34. **WebRatio:** <http://www.webratio.com/site/content/es/home> 2015 [↑](#footnote-ref-34)
35. **No Magic:** <http://www.nomagic.com/products/magicdraw.html> 2015 [↑](#footnote-ref-35)
36. **Eclipse Modelling Framwwork:** <https://www.eclipse.org/modeling/emf> 2015 [↑](#footnote-ref-36)
37. **Acceleo:** <https://eclipse.org/acceleo> 2015 [↑](#footnote-ref-37)
38. **Ecore:** Metamodelo nativo que forma parte del core del EMF para describir a los modelos [↑](#footnote-ref-38)
39. **jQuery UI 1.11 API Documentation:** <http://api.jqueryui.com/> 2015 [↑](#footnote-ref-39)
40. **jQuery Validation Plugin:** <http://jqueryvalidation.org/> 2015 [↑](#footnote-ref-40)