

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.
Ms. Nathalie Aquino.**

**Asunción
2015**

CAPITULO 1

INTRODUCCIÓN

A) CONTEXTO Y MOTIVACIONES

Hoy en día 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 [1]. 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 Applications* (RIA). Estas representan todo un desafío para la ingeniería Web, ya que las RIA han dado un cambio radical en la manera en que se comportan, desarrollan y despliegan este tipo de aplicaciones, ofreciendo mejoras substanciales con respecto a las aplicaciones Web tradicionales. Las RIA presentan nuevas características referentes a la comunicación, la distribución de los datos y la computación en el lado cliente, acompañadas de interfaces mucho más interactivas, en donde el usuario en ocasiones, no distingue si está utilizando la aplicación *online* o *offline*. Con estos avances propuestos por RIA, muchas de las metodologías Web tradicionales basadas en la Web 1.0, tales como; WebML[26], UWE[15], OOH[30], OOHD[21] u OOWS[4], han tenido que evolucionar de cierta forma, agregando nuevos modelos o extendiendo los existentes, para dar cobertura a las diversas características sofisticadas propuestas por RIA. Muchas de las metodologías citadas han logrado una notable evolución en su afán de mantenerse vigentes con los avances propuestos por las RIA, sin embargo en la actualidad, ninguna de ellas, logra satisfacer todas las nuevas funcionalidades [10][17] [7].

B) PROBLEMÁTICA

Teniendo en cuenta las limitaciones de las metodologías existentes para la cobertura de las RIA, resulta necesario crear nuevas metodologías de desarrollo Web o bien extender a las actuales para satisfacer las nuevas características impuestas por la tendencia actual. Con la idea de que los modelos de la metodología a utilizar en este trabajo de fin de carrera estén basados en estándares aceptados en la comunidad web (como UML) y a la vez puedan ser desplegados en diversas herramientas *Case* de modelado (libres o licenciadas), se ha identificado el hecho de que solo OOH y UWE poseen tales características. Sin embargo, tanto en OOH como en UWE, las soluciones RIA propuestas en sus modelos poseen detalles de alguna arquitectura destino en particular, lo que conlleva a que sus modelos no sean totalmente independientes de la plataforma. Con ese propósito, un nuevo enfoque para el desarrollo de aplicaciones Web basado en modelos y

fundamentado en los principios propuestos por la OMG¹, se ha propuesto en el DEI². Este enfoque está basado en los estándares MDA³ y ofrece un esquema de modelado en capas para la separación de conceptos. Dicho enfoque se denomina MoWebA[19][18], 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. En MoWebA es posible llevar a cabo extensiones a sus metamodelos para cubrir nuevas características, lo cual la hace adaptable a los cambios actuales.

C) OBJETIVOS GENERALES Y ESPECÍFICOS

El objetivo de este trabajo de fin de carrera se enmarca en la idea de efectuar extensiones a la propuesta Web MoWebA con respecto a la capa de Presentación, con el fin de abarcar a algunas de las principales características de las RIA.

- Analizar las diferentes propuestas metodológicas para el desarrollo de aplicaciones web, que están basadas en MDA, y que tienen la posibilidad de extenderse para permitir disponer de características RIA, enfocándose principalmente en el aspecto de la presentación enriquecida de las páginas.
- Proponer un metamodelo y reglas de transformación para aplicaciones web que permitan incorporar características RIA a nivel de presentación en la metodología MoWebA.
- Realizar un análisis crítico de la propuesta, a partir de una ilustración.

D) ESTRUCTURA DE TRABAJO

- En el capítulo 2 se definen primeramente las RIA, presentando sus principales características y los diversos enfoques existentes para la implementación de las mismas. También se presentan algunos elementos enriquecidos (*widgets*) de uso común para las interfaces de usuario.
- En el capítulo 3 se presenta el estado del arte de las metodologías de desarrollo basada en modelos MDD que dan cobertura a características de RIA. Seguidamente se presenta la aproximación de desarrollo Web MoWebA.
- En el capítulo 4 se propone una extensión a los metamodelos de Contenido y Estructura de MoWebA y se presenta una propuesta de transformación de modelo a texto (M2T) para la plataforma destino *jQueryUI*⁴ y *jQuery Validation Plugin*⁵, para cubrir algunas características RIA de las presentaciones enriquecidas y de la lógica de negocios en el lado cliente.
- En el capítulo 5 se evalúa la extensión a MoWebA por medio de una ilustración.
- En el capítulo 6 se finaliza el trabajo con un análisis de los resultados obtenidos elaborando la conclusión y los posibles trabajos futuros.

¹ Object Management Group: <http://www.omg.org/> 2015

² Departamento de Ingeniería Electrónica e Informática de la Universidad Católica Nuestra Señora de la Asunción

³ Model Driven Architecture: <http://www.omg.org/mda/> 2015

⁴ jQuery UI 1.11 API Documentation: <http://api.jqueryui.com/> 2015

⁵ jQuery Validation Plugin: <http://jqueryvalidation.org/> 2015

CAPÍTULO 2

MARCO TEÓRICO DE LAS RICH INTERNET APPLICATIONS

En este capítulo se presentarán algunas definiciones de las RIA como así también, sus principales características, tales como: el almacenamiento de datos, la lógica de negocios en el cliente, la comunicación mejorada entre el cliente y el servidor, y a las presentaciones enriquecidas. Seguidamente se dará pie a las diferentes tecnologías y herramientas para implementar a las RIA, para finalmente presentar a algunos de los *widgets* (elementos de interfaz interactivos) más utilizados en la comunidad Web para el front-end de las aplicaciones.

2.1 LAS RICH INTERNET APPLICATIONS (RIA)

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 para ir superando limitaciones de las aplicaciones web tradicionales en cuanto a la usabilidad e interactividad que ofrecen sus interfaces de usuario. En la Web 1.0, la comunicación síncrona existente entre el cliente y el servidor obliga a que por cada acceso a un enlace, el cliente deba esperar la respuesta del servidor, y una vez obtenida la respuesta, el cliente deba recargar la página completamente para actualizar una simple porción de página. Esto da lugar a retardos en el despliegue de las páginas en el cliente presentando de esta forma una interfaz poco interactiva. Con este mecanismo de comunicación, el cliente queda ocioso la mayor parte del tiempo, pudiéndose llevar a cabo actualizaciones de páginas únicamente, cuando ocurre un evento de navegación por parte de un usuario [13]. He ahí que surgen como alternativa, las denominadas Aplicaciones de Internet Enriched (Rich Internet Applications - RIA) 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 problemas relacionados a las limitaciones en cuanto a la riqueza de las interfaces, medios y contenidos de las aplicaciones [8].

Dado que las RIA poseen numerosas características innovadoras, es difícil ofrecer una definición formal que englobe todos sus atributos. Diversos autores las han caracterizado en un contexto particular y todas las descripciones presentadas a continuación resultan valederas.

“Las RIA 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 RIA 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” [17].

“En las RIA, 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 aplicación⁶”.

“Las RIA 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: 1-) el paradigma de página única; 2-) 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; 3-) la inclusión de un motor en el cliente (en la forma de máquina virtual o *plug-ins* instalados en el navegador) que administra la disposición gráfica de los elementos y la mayoría de las interacciones locales” [6].

De las descripciones anteriores, puede notarse el hecho de que en las RIA se busca que las aplicaciones tiendan a comportarse lo más similarmente a las aplicaciones de escritorio, evitando el refresco excesivo de las páginas, más precisamente, permitiendo el refresco 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 refresco 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 RIA y que ofrecen una mayor riqueza a las interfaces de usuario, como así también los elementos

⁶ Wikipedia- *Rich internet applications*: http://en.wikipedia.org/wiki/Rich_Internet_Application 2015

multimedia como audio y video *streaming*. El lado del cliente en las aplicaciones RIA 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 [6] 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 RIA presentes en esta definición.

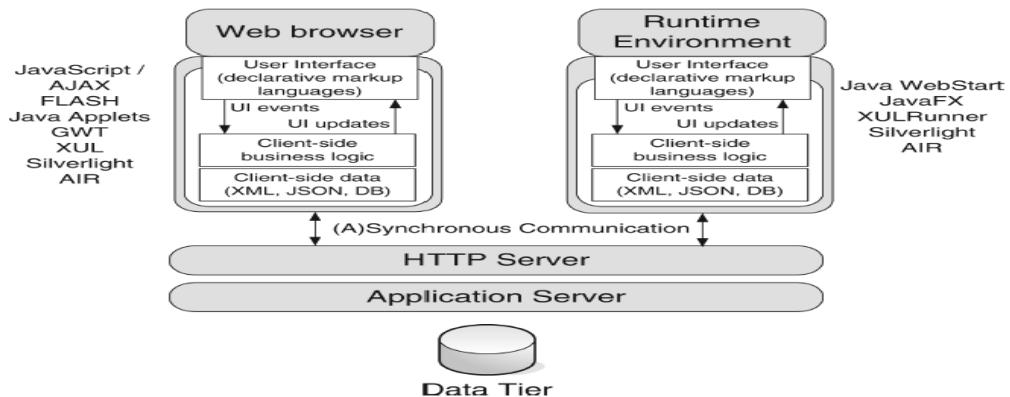


Figura 1 Arquitectura RIA [26]

Por otra parte, en [26] se describe la arquitectura de las RIA de manera general como se muestra en la Figura 1. El sistema está compuesto de un servidor de aplicaciones web y un conjunto de aplicaciones de usuario corriendo en las máquinas clientes. Estas aplicaciones son implementadas de dos formas: en un navegador web o fuera de un navegador web. En un navegador web es posible implementarla utilizando una variedad de tecnologías como *Javascript*, animaciones Flash, código interpretado en *plug-ins* y *Java applets*. Fuera de un navegador web se implementa en términos de binarios descargados desde la web e interpretados en un ambiente específico de ejecución, por ejemplo, utilizando tecnologías como *Java Web Start*⁷ y *Adobe AIR*⁸.

Las RIA hoy en día juegan un papel preponderante. Según un estudio de mercado patrocinado por la empresa Adobe en 2011, dadas las mejoras con respecto a la interfaz de usuario y al comportamiento de las aplicaciones, las RIA 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 [29]. Un estudio similar [14], presenta datos cuantitativos con referencia a cómo una aplicación con características de las RIA puede mejorar las utilidades y disminuir los costes de desarrollo en una compañía.

2.2 CARACTERÍSTICAS DE LAS RIA

⁷ Oracle: <http://www.oracle.com/technetwork/java/javase/javawebstart/index.html> 2015

⁸ Adobe AIR: <http://www.adobe.com/products/air.html> 2015

A continuación se presentan las características más distintivas de las RIA con respecto a las aplicaciones Web tradicionales, que fueron presentadas en la definición de [6] en base a una clasificación propuesta en los trabajos de [17] y [7]. 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 RIA 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
<ul style="list-style-type: none"> • 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. 	<ul style="list-style-type: none"> • 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 entre el cliente y el servidor

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 RIA 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 [26].

En la Tabla 2 se presentan algunas ventajas y desventajas de una computación distribuida de la lógica de negocios entre el cliente y el servidor.

Ventajas	Desventajas
<ul style="list-style-type: none"> • Validación de datos en vivo. • La posibilidad de utilizar la aplicación sin necesidad de una conexión (uso offline de la aplicación). 	<ul style="list-style-type: none"> • 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

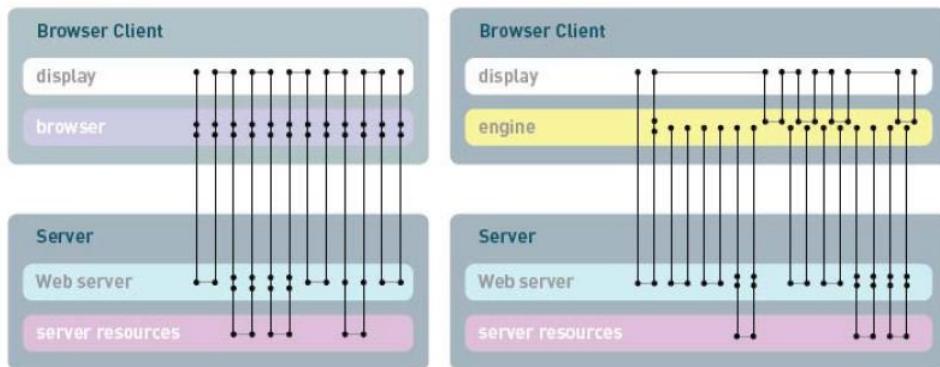


Figura 2 Comunicación síncrona versus comunicación asíncrona.[]

Con las RIA se crean mecanismos para reducir al mínimo la transferencia de los datos migrando las capas de interacción y presentación del servidor al cliente. Las RIA soportan comunicaciones asíncronas entre el cliente y el servidor para la distribución de objetos de dominio, datos y la computación. En la Figura 2 se puede ver una comparativa con respecto a la comunicación entre los pares cliente y servidor, de las aplicaciones de la web 1.0 (lado izquierdo) y las actuales basadas en RIA (lado derecho). 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 llevada a cabo de una manera síncrona, en donde un evento del usuario es necesariamente el elemento disparador de una solicitud al servidor. Con las RIA, 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 *plug-in*) instalado en el cliente, permiten 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
<ul style="list-style-type: none"> • Es posible llevar a cabo el refresco parcial de las páginas, abarcando solamente las zonas de interés. • Se mejora la interacción del usuario con la aplicación. • <i>Server-push</i>⁹. 	<ul style="list-style-type: none"> • Se incrementan los esfuerzos de desarrollo de las aplicaciones. • El <i>testing</i> 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 con la inclusión de *widgets* interactivos. Los *widgets* son micro programas empotrados dentro de las páginas web y son administrados 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 RIA. 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 refresco 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
<ul style="list-style-type: none"> • 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. 	<ul style="list-style-type: none"> • 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 RIA 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 de las aplicaciones. Por lo tanto, esta característica resulta ser percibida en primera instancia por parte de los usuarios finales. He ahí su importancia y el por qué son numerosos los *frameworks* de desarrollo existentes en la actualidad que los contemplan.

Dada la amplitud que conllevan las RIA en lo que concierne a características y funcionalidades, y con el afán de mostrar algunas de sus bondades principales de una manera concreta, este trabajo

⁹ Describe un estilo de comunicación sobre Internet donde la petición de una transacción se origina en el servidor.

de fin de carrera se enfocar principalmente en los aspectos concernientes a presentaciones enriquecidas.

2.3 TECNOLOGÍAS PARA LA IMPLEMENTACIÓN DE LAS RIA

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

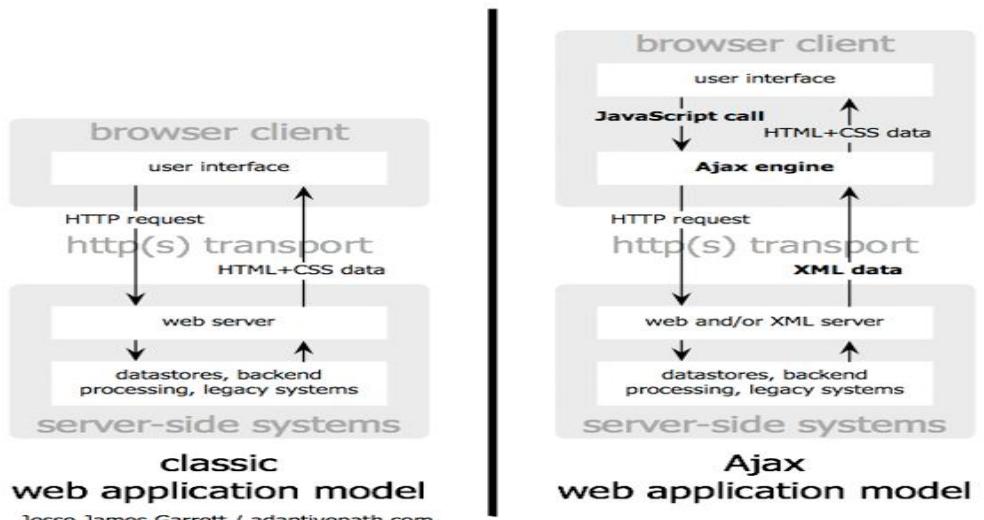


Figura 3 A la izquierda el Modelo de aplicación web clásico. A la derecha, el modelo de aplicación web Ajax.[\[13\]](#)

Basadas en Javascript: en este caso, la lógica del lado cliente está implementada en *Javascript* (el enfoque también es conocido como "Ajax", *Asynchronous Javascript* y *XML* [13] y las interfaces de usuario se basan en una combinación de *HTML* y *CSS*.

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*. El manejo de las interacciones entre la interfaz de usuario y el motor Ajax, que representan a la lógica de negocio en el lado del cliente, son implementadas por medio de *Javascript*.

El motor Ajax gestiona la comunicación entre el cliente y servidor, por medio de solicitudes *HTTP* o *HTTPS*, obteniendo las respuestas del lado servidor, interpretando los datos utilizando 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 este caso, 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 entornos de ejecución:** en este caso, 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 RIA 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 RIA	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 <i>plug-ins</i> <td>Si</td> <td>Si</td>	Si	Si
Basados en ambientes en tiempo de ejecución	Si	Si	Si	Si

Tabla 5 Tecnología cliente vs Características de las RIA

Las tecnologías basadas en *Javascript* son las más populares y más ampliamente adoptadas para el desarrollo de RIA. Sus 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). Algunas de las razones por las cuales las tecnologías *Javascript* son las más utilizadas son: 1-) Ajax es asumido por muchos desarrolladores como el conjunto más abierto y estándar de tecnologías y la más cercana a la especificación *HTML5*; 2-) no requiere acciones administrativas (por ejemplo, la instalación de software) de los usuarios, 3-) se pueden combinar fácilmente con *plug-ins* que se construyen para superar sus limitaciones.

2.4 HERRAMIENTAS PARA EL DESARROLLO DE LAS RIA

En la sección anterior se presentaron las diferentes tecnologías para el desarrollo e implementación de las RIA. Para este trabajo de fin de carrera, se optó analizar en mayor detalle las 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*. Además, esta forma de implementar las RIA es la más cercana al estándar *HTML5*.

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* 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. 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 2009¹⁰ y 2010¹¹, se presenta un análisis de los *widgets* más utilizados por las aplicaciones web. La Figura 5 presenta tales *widgets*. De todos los *widgets*

¹⁰ Designing Web Interfaces: http://designingwebinterfaces.com/essential_controls 2009

¹¹ UX BOOTH: <http://www.uxbooth.com/articles/essential-controls-for-Web-applications/> 2010

identificados, resulta interesante determinar cuáles son los más utilizados hoy en día, por lo que en el marco de este trabajo se ha realizado un análisis de portales web populares (Facebook, Gmail, Youtube y Amazon) para determinar qué *widgets* son comunes en estos sitios. El análisis determinó que los cuatro portales utilizan los siguientes *widgets*:

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

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

Autocomplete: Permite a los usuarios seleccionar texto de interés rápidamente de una lista. A medida se escribe en un campo de entrada, algunas sugerencias son presentadas al usuario, en base al patrón actual de caracteres ingresados. El usuario puede elegir una de las palabras sugeridas en un momento dado o bien seguir ingresando caracteres refinar la búsqueda.

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

Datepicker: Permite seleccionar 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 *jQueryUI*¹² y *jQuery Validation Plugin*¹³

Estudios de mercado recientes han presentado a *jQuery*¹⁴ como la librería *Javascript* más utilizada a nivel global¹⁵. *jQuery* es de código abierto y ha tenido un crecimiento notable en términos de evolución hasta hoy en día desde su aparición en el año 2005. La librería *jQuery* propone una manera robusta y confiable para desarrollar código *Javascript*. Posee extensiones para los dominios de aplicación Web (*jQueryUI*) y móviles (*jQuery Mobile*¹⁶). En ambos dominios de aplicación, se encuentran numerosos *widgets* interactivos idóneos para las interfaces de usuario enriquecidas.



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

¹² **jQuery user interface:** <http://jqueryui.com/> 2015

¹³ **jQuery Validation Plugin:** <http://jqueryvalidation.org/> 2015

¹⁴ **jQuery:** <http://jquery.com/> 2015

¹⁵ **Usage of JavaScript libraries for websites** http://w3techs.com/technologies/overview/javascript_library/all 2015

¹⁶ **jQuery Mobile:** <http://jquerymobile.com/> 2015

De todas las características citadas en la sección anterior, *Live validation* es el único *widget* no soportado por *jQueryUI* de manera nativa. Sin embargo con *jQuery Validation Plugin* (extensión basada en *jQuery*), es factible llevar a cabo validaciones locales 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 RESUMEN DEL CAPÍTULO

En este capítulo se han visto las diversas características que ofrecen las RIA, 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 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 actualidad¹⁷. 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 considerando importantes portales web, entre los *widgets* más utilizados se encuentran los *tooltips, tabs, accordion, datepicker y autocomplete*, como así también diversas validaciones locales en los campos de entrada (por ejemplo, validaciones de tipo numérico, email, claves, etc.). Actualmente, *jQuery* es la librería *Javascript* más popular. Además, utilizando sus versiones *jQueryUI* y *JQuery Validation Plugin* se brinda cobertura a todas estas características enriquecidas.

¹⁷ List of Ajax frameworks: http://en.wikipedia.org/wiki/List_of_Ajax_frameworks 2015

CAPÍTULO 3

ENFOQUES METODOLÓGICOS MDD PARA LAS RIA

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

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

3.1 MODEL DRIVEN SOFTWARE ENGINEERING (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 [16]. 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 transformarse 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 MDD para el ciclo de desarrollo de una aplicación. MDD es un paradigma de desarrollo que utiliza a los modelos como artefactos primarios en el proceso de desarrollo. Usualmente en MDD la implementación es generada de manera automática o semiautomática a partir de los modelos. Por otra parte, MDA¹⁸ es un estándar, impulsado por el consorcio OMG (*Object Management Group*), que contiene en sí mismo a varios estándares de facto tales como UML¹⁹ (*Unified Modeling Language*), OCL²⁰ (*Object Constraint Language*), MOF²¹ (*Meta Object Facility*), QVT²² (*Query View Transformation*), entre otros. MDA promueve el desarrollo de software para diversos dominios de aplicación, como las aplicaciones para el ámbito de las finanzas, las telecomunicaciones, las aplicaciones aeroespaciales, las embebidas, etc. MDA es un subconjunto de MDD que propone estándares para cada paso en el proceso de desarrollo de las aplicaciones. Utiliza un esquema de arquitectura dividida en capas como puede apreciarse en la Figura 1. Los meta-metamodelos (M3) se expresan por medio de MOF o ECORE para el *Eclipse Modelling Framework (EMF)*. Los metamodelos (M2) de la aplicación se expresan por medio de un *General Purpose Modelling Language (GPML)* (por lo general UML) que cuenta con diversos modelos para representar los comportamientos (estáticos y dinámicos) de una aplicación en particular. La capa M2 describe los conceptos utilizados en M1 para la definición de los modelos. Finalmente el objeto del mundo real, en el ejemplo de la Figura 1 un video, se representa en M0.

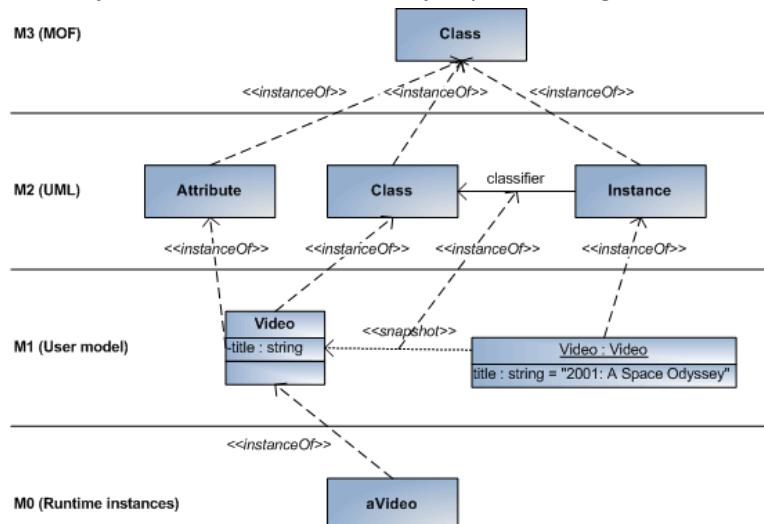


Figura 6 Arquitectura dividida en capas de MDA: Objetos del mundo real (M0), modelos (M1), metamodelos (M2) y meta-metamodelos (M3)

Las fases de desarrollo con el enfoque MDA se presentan en la Figura 7. En la primera fase se tiene el *Computation Independent Model* (CIM), que corresponde a los documentos, modelos o diagramas utilizados para la toma de requerimientos de una aplicación en particular,

¹⁸ MDA www.omg.org/mda/ 2015

¹⁹ UML: www.omg.org/spec/UML 2015

²⁰ OCL: www.omg.org/spec/OCL 2015

²¹ MOF: www.omg.org/mof/ 2015

²² QVT: www.omg.org/spec/QVT/1.1/ 2015

independientemente de cómo planean ser implementados. Representan el punto de vista del negocio de la solución.

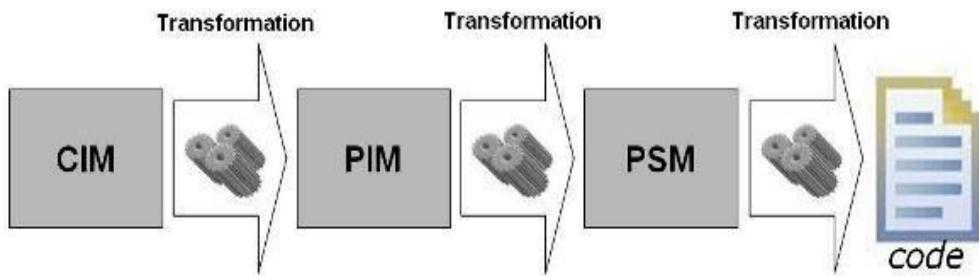


Figura 7 Cadena de transformaciones en MDA

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

3.1.2 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 (*Model Driven Engeneering*), la generación de código es el proceso de transformar modelos en código fuente.

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

¿Qué tanto código va a generarse?

La pregunta principal aquí es qué parte del código puede ser automáticamente generada a partir de los modelos. ¿Es posible llevar a cabo una generación de código parcial o total? La generación parcial de código puede implicar muchas cosas en este contexto. Primero, puede implicar que una capa (horizontal o vertical) de la aplicación sea completamente generada, mientras que otra capa

podría ser desarrollada completamente de manera manual. Más aún, una capa puede ser generada parcialmente y otras partes no cubiertas tienen que ser completadas con código manual. La generación parcial de código también puede referirse al nivel de modelado, utilizando solamente la generación de código para ciertas partes del modelo, mientras que otras partes no son manipuladas por el generador de código y tienen que ser implementadas manualmente.

¿Qué código va a generarse?

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

¿Cómo va a generarse?

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

3.1.3 Una vista de los lenguajes de transformación basados en plantillas

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

XSLT²³

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

JET - Java Emitter Template²⁴

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

Xpand²⁵

²³ XSLT: www.w3.org/TR/xslt20/ 2015

²⁴ JET: <https://projects.eclipse.org/projects/modeling.m2t.jet> 2015

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

MOFScript²⁶

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

Acceleo²⁷

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

3.3 Beneficios de los lenguajes de transformación

Los lenguajes de transformación M2T separan el código estático y dinámico utilizando el enfoque de plantillas (*plantillas*) para implementar las transformaciones M2T. Una plantilla puede ser vista como un proyecto que define elementos de texto estáticos compartidos por todos los artefactos, como así también, partes dinámicas que deben ser completadas con información específica para cada caso en particular. Por lo tanto, un *template* contiene fragmentos de texto simples para las partes estáticas y los llamados metamarcadores (*meta-markers*) para las partes dinámicas. Los metamarcadores son marcadores de posición y deben ser interpretados por un motor que procesa los *plantillas* y consulta fuentes de datos adicionales para producir las partes dinámicas. Las fuentes adicionales de datos son los modelos. En la Figura 8 se presenta el esquema tradicional de transformación basado en *plantillas*.

²⁵XPAND: <https://eclipse.org/modeling/m2t/?project=xpand> 2015

²⁶MOFScript: <https://eclipse.org/gmt/mofscript/> 2015

²⁷Acceleo: www.acceleo.org/ 2015

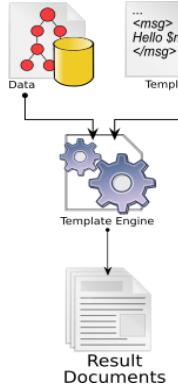


Figura 8 Plantilla, motor de plantillas y modelos de entrada para producir texto

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

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

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

3.2 PRINCIPALES ENFOQUES DE DESARROLLO WEB BASADO EN MODELOS PARA LAS RIA

En [11] y [10] se identifica la necesidad de metodologías sistemáticas para el desarrollo de las RIA y se llevan a cabo estudios presentando las diversas metodologías web existentes para ese fin. Un estudio más exhaustivo y reciente de comparativas se presenta en [7] en donde se clasifican las metodologías en las siguientes categorías:

- Contribución a la investigación proveniente de la comunidad de ingeniería web, derivada de la evolución de los enfoques dirigidos por modelos concebidos para el diseño y desarrollo de aplicaciones web tradicionales. Esta categoría incluye a WebML-RIA[26], OOHDM-RIA[21], OOH4RIA[30], (UWE-R[15], Patrones con UWE[24] y UWE+RUX [12]).
- Enfoques de desarrollo sistemáticos provenientes de la comunidad de *Human Computer Interaction (HCI)*, en donde el diseño RIA es el foco principal. Las metodología RUX [20] y UsiXML[6] [5]. pertenecen a esta categoría
- Enfoques que combinan HCI y técnicas de ingeniería web: espacios interactivos con UML presentado en [25] y OOWS for RIA [4].

- d) Propuestas recientes de los vendedores de herramientas comerciales que adoptan MDD, entre ellos WebRatio, Mendix, Novulo, RUX-Tool y Thinkwise.

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

3.2.1 Extensión a OO-H (OOH4RIA)

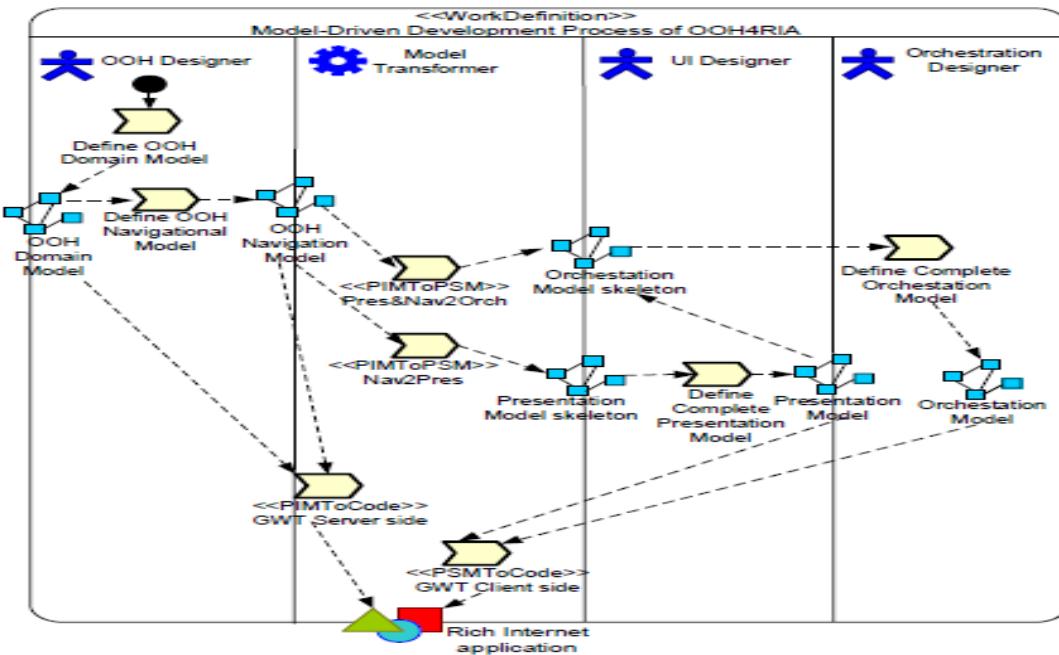


Figura 9 Representación del proceso MDD para OOH-RIA

OO-H (*Object Oriented Hypermedia*) [9] 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 Figura 4 muestra una representación del proceso MDD con las definiciones de modelos y transformaciones que permiten obtener la implementación correspondiente a las RIA, como así también, los actores que participan en el ciclo de desarrollo. OOH4RIA, propone un metamodelo de presentación definido con abstracciones de bajo nivel, donde los elementos principales son representados por los

widgets proveídos por una plataforma específica, en este caso *Google Web Toolkit (GWT)*²⁸. Este metamodelo permite la especificación de los aspectos estructurales de las RIA. Los *widgets* pueden ser combinados, extendidos, adaptados y enlazados a otros modelos. Se genera el código de la aplicación tanto para el lado cliente como para el lado servidor.

3.2.2 Extensiones RIA a UWE (UWE-R)

El enfoque UWE (*UML-based Web Engineering*) [22][23] 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 Figura 10, 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*.

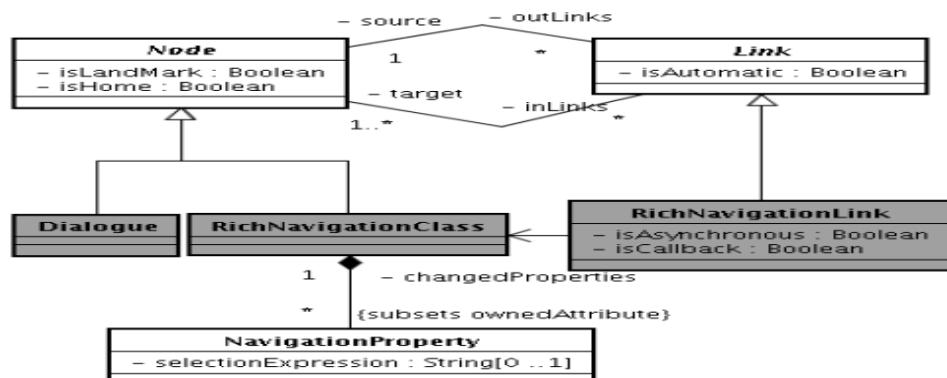


Figura 10 Extensiones a las metaclases Nodo y Enlace.

A nivel de presentación, se agregan algunas metaclases para expresar la riqueza de las RIA con respecto al aspecto de la interfaz de usuario, como puede verse en la Figura 11.

²⁸ Google Web Toolkit: <http://www.gwtproject.org/> 2015

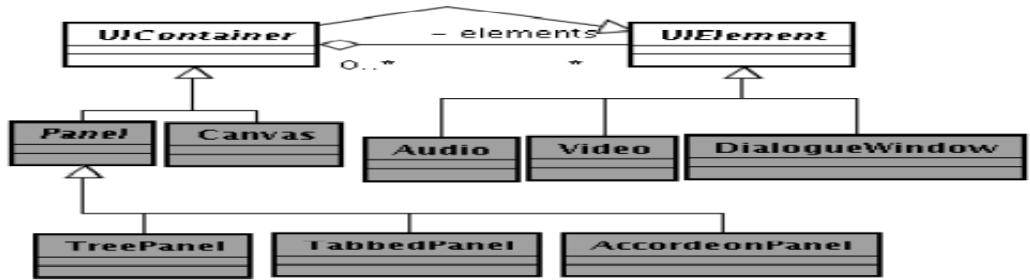


Figura 11 Extensiones al metamodelo de Presentación en UWE-R.

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

3.2.3UWE combinada con la herramienta RUX

La metodología UWE también puede combinarse con el método RUX [12]. La metodología RUX es un enfoque dirigido por modelos para el enriquecimiento de las interfaces de usuario. Puede ser utilizada en el tope de muchas metodologías de modelado web. En este enfoque, UWE es utilizado para especificar el contenido, navegación y proceso de negocio de una aplicación Web, y la metodología RUX se emplea sobre estos modelos para adicionar capacidades enriquecidas a la interfaz de usuario. En esta propuesta se busca construir el puente entre ambos enfoques, definiendo reglas de transformación entre sus respectivos metamodelos. En otros términos, se extienden las reglas de generación de UWE de manera a obtener la conexión con la metodología RUX automáticamente.

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

La interfaz final contiene la información final para la generación de código de la interfaz de usuario, lo cual es específico para un dispositivo o un grupo de dispositivos y para una plataforma de desarrollo RIA tal como Flex, Ajax o OpenLaszlo²⁹.

3.2.4 UWE con patrones

UWE puede extenderse por medio de patrones [24]. Los patrones RIA describen la interacción, la operación y la presentación de un *widget*, en donde la interacción es el disparador del patrón RIA (por ejemplo, el movimiento del mouse, presionar una tecla o un evento temporal). Como resultado de la interacción, una variedad de operaciones pueden ser llevadas a cabo, tales como validaciones, búsquedas y refrescados de página. Finalmente el resultado implica una actualización en la interfaz de usuario.

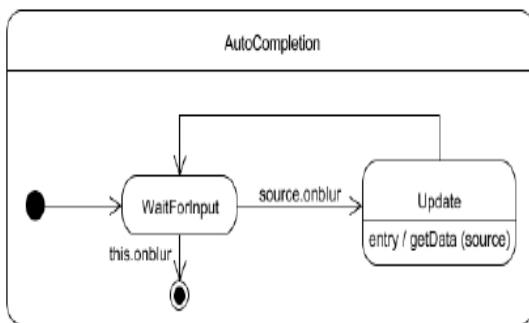


Figura 13 Patrón Autocomplete en UWE

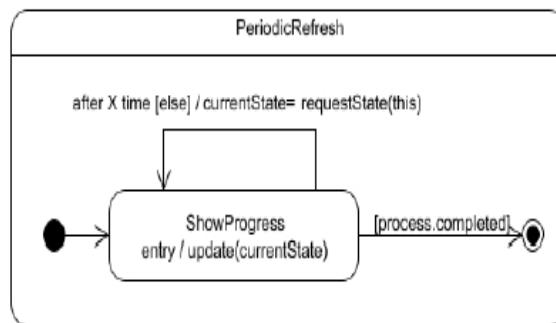


Figura 12 Patrón Periodic Refresh en UWE

Este enfoque consiste en el uso de modelos que representan *widgets* RIA, y la inclusión de estos modelos dentro de metodologías de desarrollo web existentes. Cada *widget* es modelado por medio de máquinas de estados que representan la característica RIA deseada. En la **¡Error! No se encuentra el origen de la referencia**. Figura 8 y la Figura 7 se muestran los patrones definidos en el trabajo de Koch[24]. .

3.2.5 Patrones en OOWS

La principal contribución de Valverde, es un modelo de interacción para especificar la nueva semántica para hacer frente al desarrollo basado en modelos RIA [4]. 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)*³⁰. 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/collapse de texto. En la Figura 14 se puede analizar el proceso de desarrollo para esta propuesta.

²⁹OpenLaszlo: <http://www.openlaszlo.org/> 2015

³⁰Interaction Design Foundation: http://www.interaction-design.org/encyclopedia/human_computer_interaction_hci.html 2015

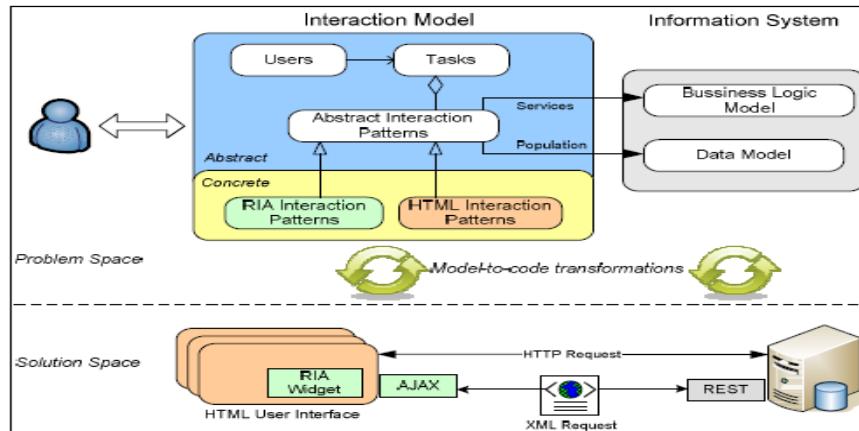


Figura 14 Un resumen del enfoque MDD con patrones para OOWS de Valverde y Pastor

La Tabla 7 que se muestra a continuación presenta un resumen de las propuestas, indicando las características RIA contempladas por las mismas que fueron expuestas en el Capítulo 2 del libro.

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

Tabla 6 Metodologías web y sus alcances para RIA

En el análisis de la Tabla 6, se nota que la metodología que más características de las RIA abarca es WebML, con la salvedad que utiliza herramientas propietarias para su modelado³¹, se basa en un DSL gráfico propio, no utiliza UML y no cubre widgets. Con respecto a la característica de presentaciones enriquecidas, que es la que concierne a este trabajo de tesis, la metodología RUX y la combinación de UWE+RUX son las que ofrecen cobertura completa a diferencia de las otras metodologías. Sin embargo, RUX no es precisamente una metodología, sino más bien una herramienta propietaria que sirve para enriquecer con características de las RIA a las metodologías web. UsiXML ofrece una metodología estándar bastante completa que utiliza una serie iterativa de transformaciones XSLT (*Extensible Stylesheet Language Transformations*) para obtener la interfaz de usuario final para una plataforma destino a partir de una interfaz abstracta, definida previamente, pero está abocada específicamente al desarrollo de interfaces y no es una metodología que abarque todo el ciclo de vida de una aplicación web.

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

3.3 LA APROXIMACIÓN MOWEBA (MODEL ORIENTED WEB APPROACH)

MoWebA [18][19] es una propuesta creada en el DEI (Departamento de Electrónica e Informática) que adopta los principios de MDA. En la Figura 10 se muestran las dimensiones de MoWebA. Como puede observarse, consta de fases, niveles y aspectos, que se van describiendo a continuación.

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

1. **Modelado del problema:** incluye al CIM (*Computation Independent Model*), orientado al modelado de los requisitos funcionales, y al PIM (*Platform Independent Model*), orientado al modelado del problema sin considerar aspectos de la arquitectura o plataforma. A partir de aquí es posible llevar a cabo transformaciones para obtener los modelos específicos de la plataforma de manera semi-automática por medio de reglas de transformación.
2. **Modelado de la solución:** incluye al ASM (*Architectural Specific Model*) y al PSM (*Platform Specific Model*). Es en esta fase en donde todos los detalles de la arquitectura y plataforma destino se definen, permitiendo generar a partir de aquí, el código de la aplicación de manera automática. En MoWebA se independiza esta fase, y esto hace que sea bastante prometedora para la implementación de las RIA, debido a que existen numerosas plataformas destino para desplegarlas. En las aproximaciones estudiadas, por lo general las extensiones RIA son definidas en

³¹ **WebRatio:** <http://www.webratio.com/site/content/es/home> 2015

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.

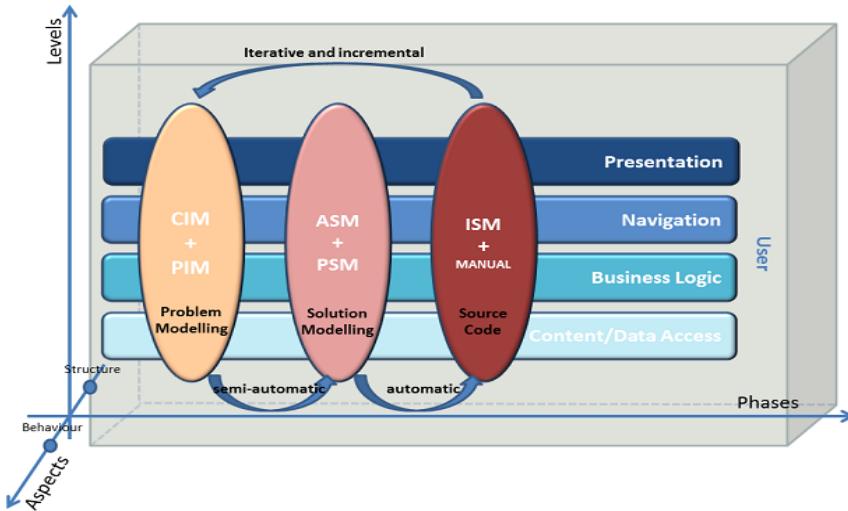


Figura 15 Fases, niveles y aspectos en el desarrollo de MOWEBA.

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

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

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

3.3.1 La capa de presentación de MoWebA

La capa de presentación de MoWeba abarca a los metamodelos de contenido y estructura (ver Figura 16). En el metamodelo de contenido (*Content*) se tienen los diversos elementos de interfaz (*uiElements*) correspondientes a la web 1.0. Entre estos elementos están los *textInput*; los enlaces, que podrían corresponder a navegaciones internas de la aplicación (*anchor*), o bien a navegaciones externas (*externalLink*); los *button*; los elementos del tipo selección que

corresponden a los *choice* y a los *dropBox*; los *text*, para texto plano en las páginas; *htmlText*, para el despliegue de cualquier texto HTML; y el elemento del tipo *multimedia* para audio y video. Cada uno de estos elementos cuenta con sus respectivos *atributos* para identificar a sus propiedades intrínsecas. El *compositeUIElement* contiene a los distintos *uiElements* y en él pueden definirse condiciones de tipo *orderBy* y *groupBy*, en caso que sea necesario obtener datos del modelo de dominio. El elemento de interfaz *form* extiende al *compositeUIElement*, permitiendo definir a los distintos *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*.

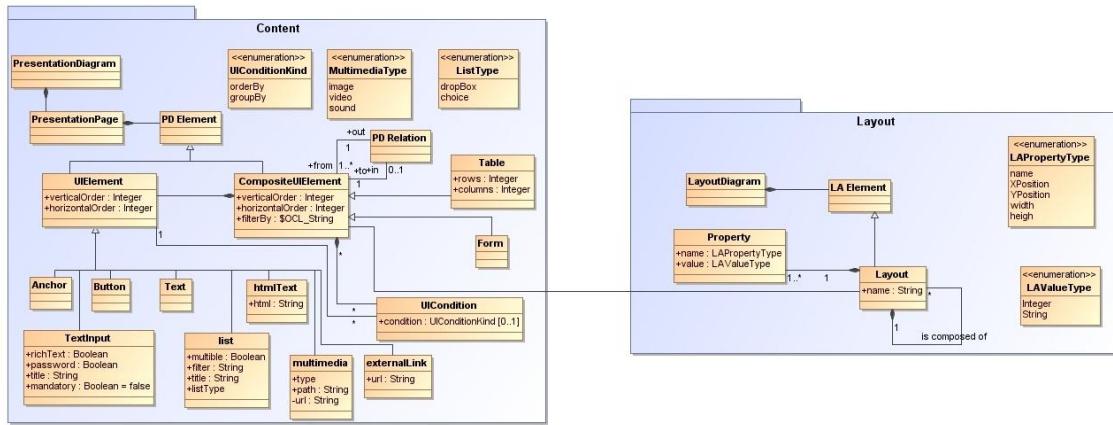


Figura 16 Metamodelo de Contenido y Estructura de MoWebA

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

Los metamodelos *Content* y *Layout* definen la sintaxis abstracta de la capa de presentación de MoWebA por medio del estándar MOF (*Meta Object Facility*). La sintaxis concreta de MoWebA es llevada a cabo por medio de UML, utilizando la técnica de perfil (*profiling*), que permite agregar a UML los diversos estereotipos (*stereotypes*) y valores etiquetados (*tagged values*) propios de MoWebA. Los perfiles *Content* y *Layout* permiten definir los PIM de presentación de una aplicación modelada con MoWebA. Los perfiles de Contenido y Estructura se presentan en la Figura 17 y en la Figura 18.

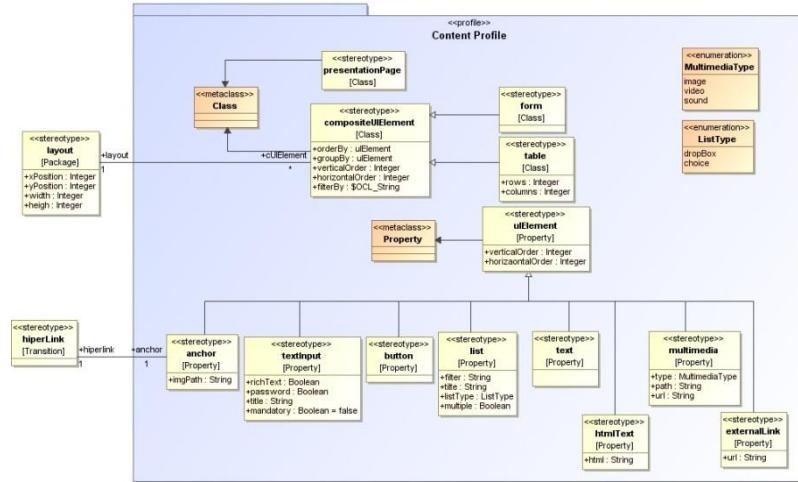


Figura 17 Perfil de Contenido de MoWebA

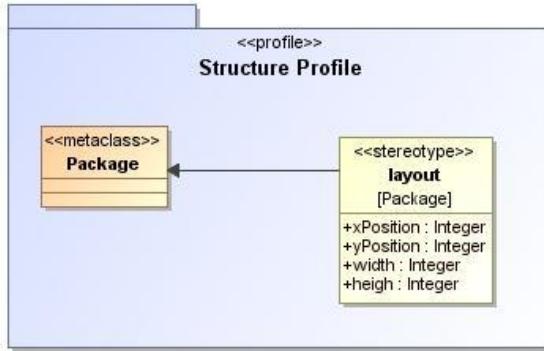


Figura 18 Perfil de estructura de MoWebA

A modo de ejemplo, se presenta en la Figura 19, el PIM correspondiente a la presentación de una aplicación con MoWebA, en la que se solicita el ingreso de datos personales, utilizando para el modelado, el perfil de Contenido y el de Estructura. En la Figura 20, se presenta la interfaz de usuario obtenida a partir del PIM de la Figura 19.

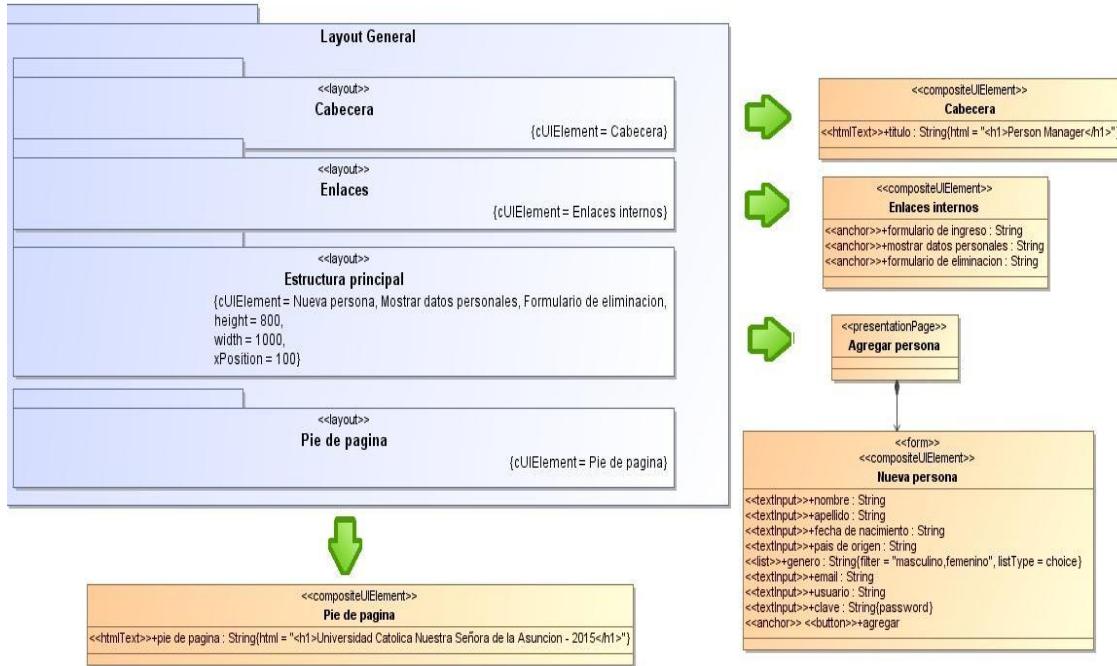


Figura 19 PIM modelado con el perfil de contenido de MoWebA

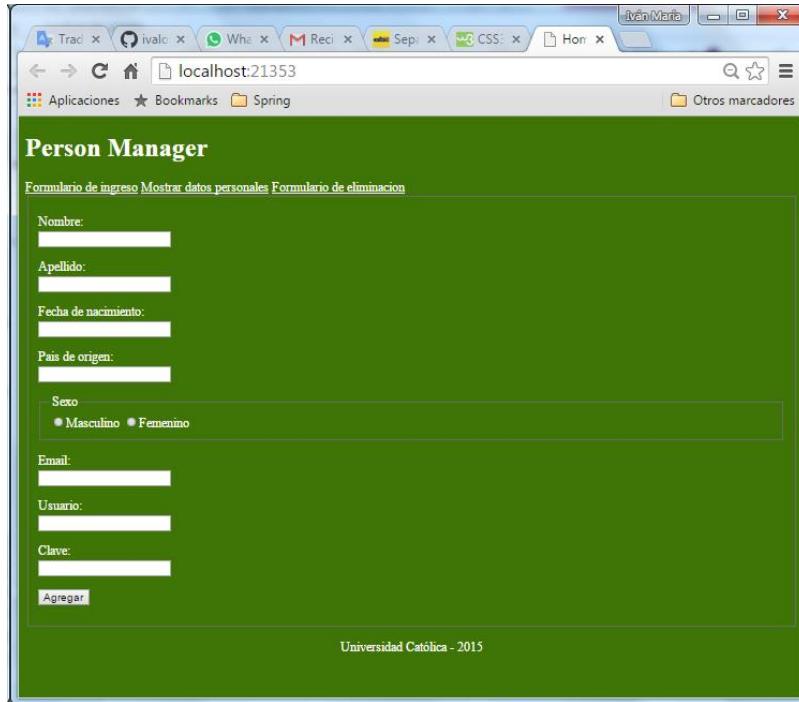


Figura 20 Interfaz obtenida a partir del PIM presentado en la Figura 14

3.3.2 El enfoque utilizado con MoWebA para la generación de interfaces

La Figura 21 representa el proceso para el modelado y generación de interfaces con MoWebA. Primeramente, se modelan los PIM que representan a una aplicación en particular utilizando

distintos perfiles UML de MoWebA. Estos perfiles representan extensiones a UML para agregar características específicas de MoWebA a los metamodelos, para que de esta forma sea posible representar la sintaxis concreta del DSL (*Domain Specific Language*). 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 MagicDraw³². Posteriormente, tanto los PIM como los perfiles son exportados al formato XMI del EMF³³. Esto de por sí es llevado a cabo a fines de tener compatibilidad con la herramienta de transformación M2T Acceleo³⁴, que toma como entrada modelos UML que están basados en el metamodelo Ecore³⁵.

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.

³² No Magic: <http://www.nomagic.com/products/magicdraw.html> 2015

³³ Eclipse Modelling Framework: <https://www.eclipse.org/modeling/emf> 2015

³⁴ Acceleo: <https://eclipse.org/acceleo> 2015

³⁵ Ecore: Metamodelo nativo que forma parte del core del EMF para describir a los modelos

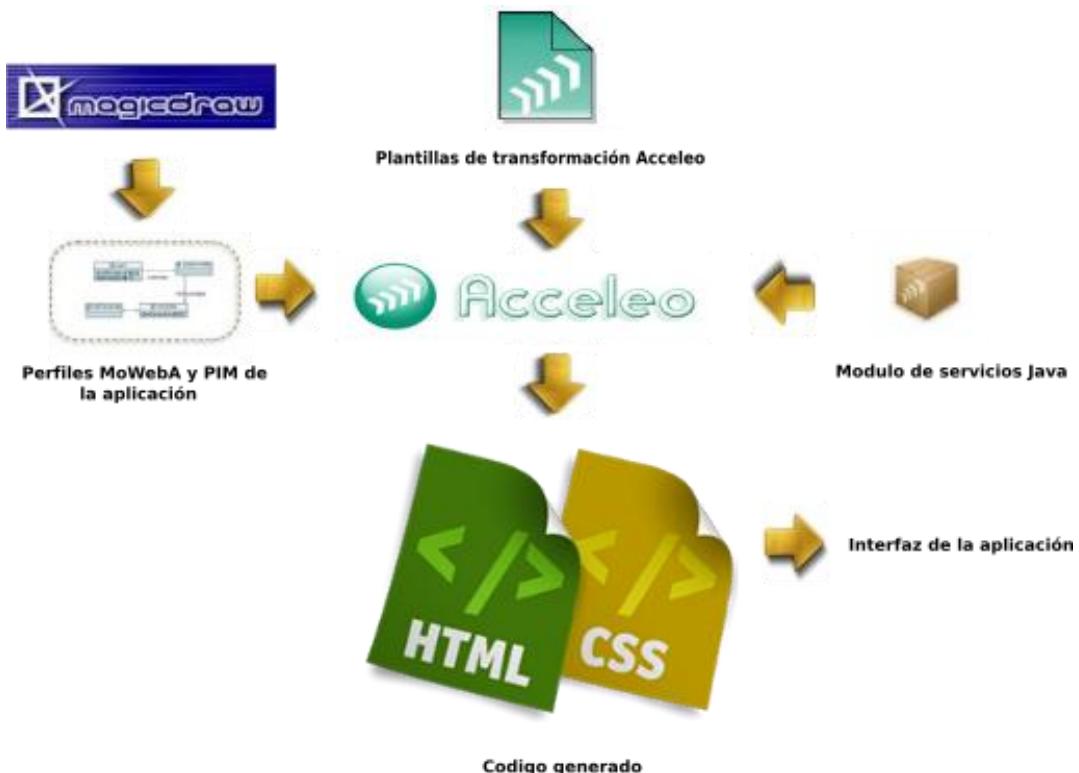


Figura 21 Fases para la generación de interfaces con MoWebA

3.7 RESUMEN DEL CAPÍTULO

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

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

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 RIA 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 RIA, 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 RIA 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 las 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* y el *tooltip*) serán presentados en MoWebA como *richAccordion*, *richTabs*, *richAutoSuggest*, *richDatePicker*, *richToolTip*. El *live validation*, describe diversas validaciones a los campos de entrada y se presentan en MoWebA como *richMinLength*, *richMaxLength*, *richOnlyDigits*, *richConfirmPass* y *richEmail* respectivamente.

4.1 EL ENFOQUE UTILIZADO CON MOWEBA PARA LA GENERACIÓN DE INTERFACES ENRIQUECIDAS

La Figura 22 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 capítulo anterior. Sin embargo, tanto el metamodelo de contenido de MoWebA como también su perfil, han sido extendidos con nuevos elementos de interfaz de usuario que son típicos de las RIA. También las plantillas de transformación, han sido adaptadas para generar el código correspondiente a cada uno de los nuevos elementos de interfaz RIA 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 diversos tipos de validación local de campos. De igual manera, que en su forma original, es posible generar 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.

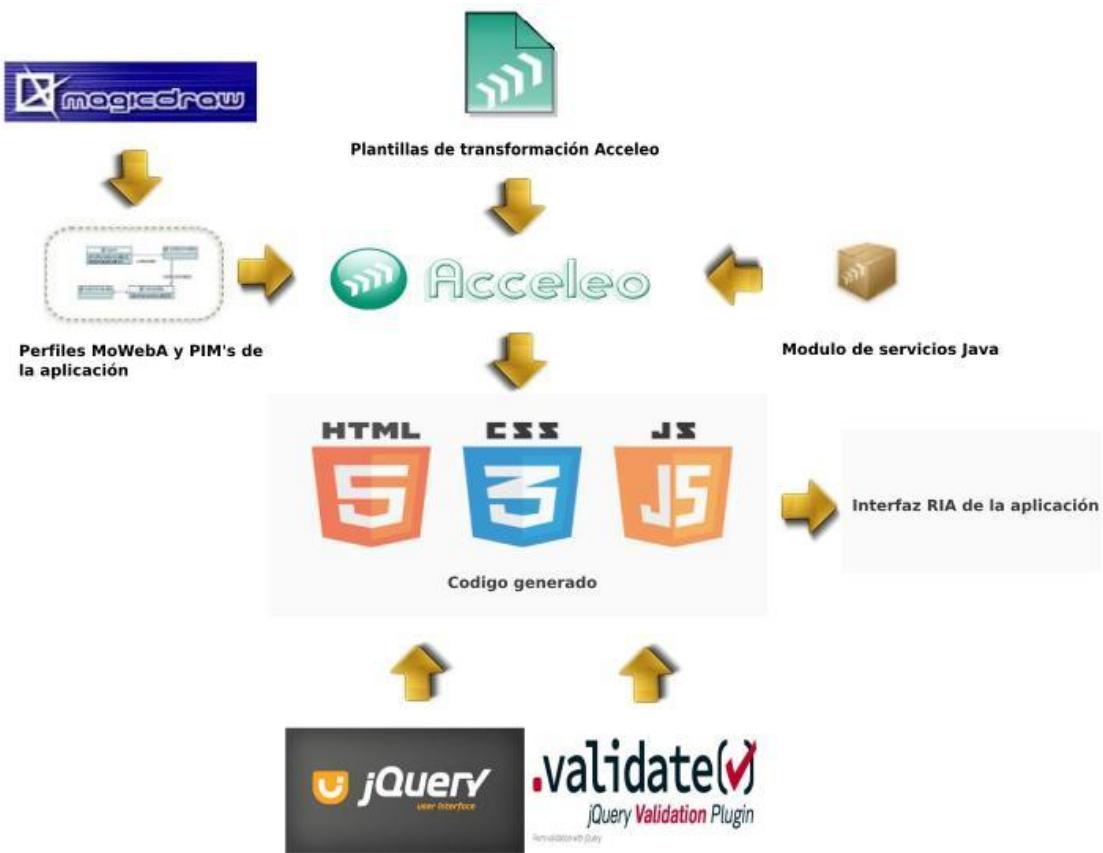


Figura 22 Fases de desarrollo para la propuesta de extensión a MoWebA

4.2 – EXTENSIONES A LA CAPA DE PRESENTACIÓN DE MOWEBA.

El objetivo de este trabajo de tesis es agregar nuevos elementos enriquecidos al metamodelo y perfil de *Contenido* de MoWebA, 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. También se extenderá el perfil de *Contenido*, que permitirá el modelado de la sintaxis concreta, precisamente los diversos modelos PIM representados con diagramas UML.

Las nuevas extensiones propuestas a los metamodelos de *Contenido* y *Estructura* de MoWebA se presentan en la Figura 23. En ellos se despliegan los diversos elementos que permiten representar una interfaz de usuario enriquecida. Los diferentes elementos tanto del metamodelo y perfil de *Contenido* como el de *Estructura*, fueron catalogados en diferentes colores para diferenciarlos de su forma original, estableciendo el color salmón para las clases que no han sufrido ningún cambio con respecto a la versión original de MoWebA, color celeste para las clases originales de MoWebA que han sufrido modificaciones de agregado, modificación o eliminación de propiedades y color verde para las clases y enumeraciones nuevas.

4.2.1 El metamodelo de Contenido extendido

Primeramente en el metamodelo de *Contenido*, se estableció una jerarquía entre los elementos compuestos (*CompositeUIElement*) y los elementos simples o elementos hoja (*UIElements*), aplicando el patrón *composite*, que es de uso común en el mundo de la ingeniería de software, principalmente cuando se desea desarrollar soluciones generales. El patrón *Composite* permite crear una jerarquía de elementos anidados unos dentro de otros. Cada elemento permite alojar una colección de elementos del mismo tipo, hasta llegar a los elementos “reales” que se corresponden con los nodos “Hoja” del árbol [3]. Para el caso de MoWebA, cada *CompositeUIElement* puede contener uno o más elementos *PD Element* que a la vez pueden ser compuestos (*compositeUIElement*) o simples u hojas (*UIElement*). El *PD Element* que corresponde a una clase padre abstracta, contiene las propiedades *horizontalOrder* y *verticalOrder* para indicar el orden horizontal y vertical de un elemento simple o compuesto. El *PD Element* puede acceder al modelo de datos y para ese caso, pueden establecerse cero o muchas condiciones sobre estos elementos, del tipo *order by* y *group by*, que forman parte de la clase *UICondition*.

Como un nuevo aporte al metamodelo de *Contenido* de MoWebA, se propone la clasificación de los diferentes elementos simples de interfaz (*UIElement*), 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 *UIElements* se clasifican de la siguiente forma:

- **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*, *multimedia* y *richToolTip*.

- **Elementos de entrada (*inputElements*):** Comprende a los elementos de interfaz enriquecidos y tradicionales encargadas de obtener una entrada desde la interfaz de usuario. En esta categoría se engloba a los elementos *textInputs*, *list*, *richAutoSuggest*, *richDatePicker*, *password* y *richEmail*.
- **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*.

Formando parte también de la extensión, los *CompositeUIElement*, pueden o no tener *Panels* asociados y los *Panels* pueden estar asociados a uno o muchos *CompositeUIElement*. Los *Panels* pueden formar parte de un *RichAccordion* o un *RichTabs*, y tanto el *RichAccordion* como el *RichTabs* pueden contener uno o muchos *Panels*. Cada uno de estos *Panels*, permite aglomerar a uno o muchos elementos de interfaz *CompositeUIElement*. Cada *Panels*, puede formar parte de un *RichAccordion* o un *RichTabs*. De manera inversa un *RichAccordion* o un *RichTabs* se compone de uno o muchos *Panels*.

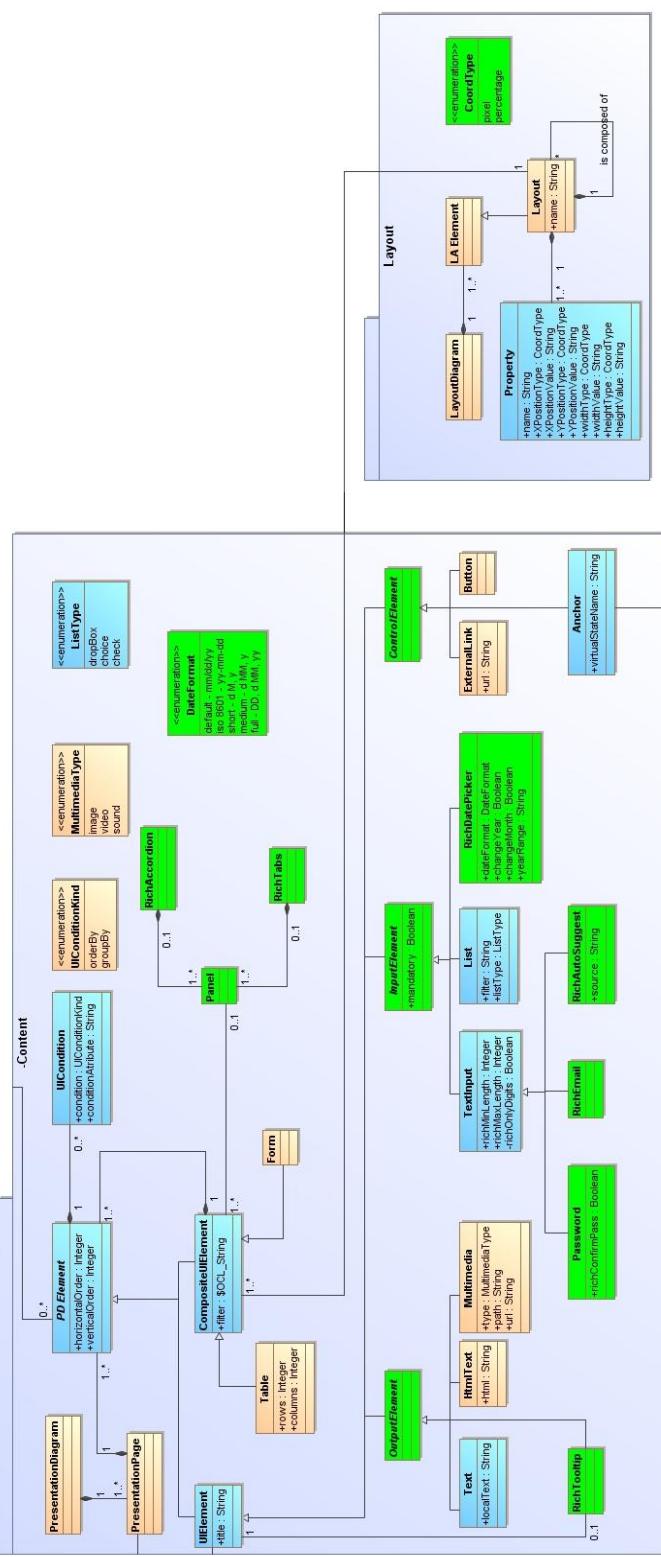


Figura 23 Metamodelo de contenido y estructura

A continuación se describen cada uno de los elementos que forman parte de la extensión al metamodelo de Contenido de MoWebA con sus respectivas propiedades.

4.2.1.1 RichAutoSuggest

Este elemento de interfaz enriquecido de entrada, contiene al atributo *source*. Este atributo tiene una doble funcionalidad. Una de ellas es permitir definir en él, un listado de palabras separadas por el carácter especial “@”, que corresponde a las palabras que serán sugeridas en el momento de ingresar uno o varios caracteres en un campo del tipo *RichAutoSuggest*. Por ejemplo, para el campo *País* de origen del tipo *RichAutoSuggest*, el atributo *source*, puede definirse como *source*=“Paraguay@Portugal@PaquistanPolonia@Peru@España@...”. La otra funcionalidad del atributo *source* permite definir en él, una ruta en la cual se aloja un archivo .xml que contiene el listado de palabras que corresponde a las sugerencias. Por ejemplo, *source* puede estar definido de la siguiente forma, *source*=“países.xml”, en donde países.xml tiene el siguiente formato:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<tags style="MEDIUM">
    <tag>
        <name>Paraguay</name>
    </tag>
    <tag>
        <name>Portugal</name>
    </tag>
    <tag>
        <name>Paquistan</name>
    </tag>
    .
    .
    .
    <tag>
        <name>...</name>
    </tag>
</tags>
```

También es posible obtener el listado de palabras desde el modelo de datos de MoWebA, estableciendo una relación de dependencia entre el elemento *RichAutoSuggest* y un *Value Object* que contiene la información necesaria de una entidad en particular. En la Figura 3 se presenta un ejemplo del widget *RichAutoSuggest*.



Figura 24 El widget RichAutoSuggest

4.2.1.2 RichDatePicker

Este elemento de interfaz enriquecido de entrada, contiene a los atributos *dateFormat*, *changeYear*, *changeMonth* y *yearRange*. El *dateFormat* corresponde a un tipo de dato enumerable que permite seleccionar cinco formatos de fecha distintos que pueden ser:

- * **Default - mm/dd/yy:** Este formato es el valor por omisión de numerosas librerías Javascript. Por ejemplo, 06/08/2015
- * **ISO 8601 - yy-mm-dd:** Este formato es el ISO 8601 para el establecimiento de fechas. Por ejemplo, 2015-06-08
- * **Short - d M, y:** Este es un formato de fecha corta. Por ejemplo, 8 Jun, 15
- * **Medium - d MM:** Este es un formato de fecha mediana. Por ejemplo, 8 June, 15
- * **Full - DD, d MM, yy:** Este es un formato de definición de fecha completa. Por ejemplo, Monday, 8 June, 2015

El atributo *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. En la Figura 4 se presenta el widget *RichDatePicker* con el formato de fecha *Default - mm/dd/yy*.



Figura 25 El widget *RichDatePicker*

4.2.1.3 RichToolTip

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 26 El widget *RichToolTip*

Al definirse este elemento 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. En la Figura 5 se presenta el widget *RichToolTip*, desplegando un mensaje al posicionar el puntero del mouse sobre el campo Contraseña del tipo *Password*.

4.2.1.4 Live Validation

El *Live Validation* es un conjunto de extensiones que permite llevar a cabo validaciones locales a diversos elementos pertenecientes a un formulario. Estas validaciones pueden llevarse a cabo a diversos elementos de entrada, como a los del tipo *TextInput*, *RichEmail*, *Password* y a los elementos del tipo *List*, *choice* y *check*.

Para los campos del tipo *TextInput*, *Password* y *RichEmail*, es posible establecer la cantidad mínima de caracteres que puede ingresarse, por medio del atributo entero *minLength*. El atributo *minLength* resulta ideal para campos del tipo *Password* para el establecimiento de un nivel de seguridad en las contraseñas. De manera similar, el atributo *maxLength* permite establecer la cantidad máxima de caracteres que es posible ingresar en estos campos, para evitar desbordamientos. El campo *TextInput*, independientemente a *Password* y *RichEmail*, posee el atributo privado *digits*, que establece que el campo de entrada debe tener estrictamente valores numéricos del cero al nueve. El campo del tipo *Password* posee el atributo booleano *confirmPass*,

para el caso en el que se necesite crear otro campo de entrada del tipo *Password* para la confirmación de contraseña.

El atributo booleano *mandatory* de la clase abstracta *InputElement*, puede activarse para todos los campos que heredan de ella. Para el caso de los campos, *TextInput*, *Password*, *RichEmail*, *RichDatePicker* y *RichAutoSuggest*, el atributo *mandatory* indica que estos campos no pueden quedar vacíos. Para el campo del tipo *List*, que puede ser un *dropBox*, *choice* o *check*, al activar el atributo *mandatory*, implica que al menos una de las opciones de un *dropBox*, *choice* o *check*, debe ser seleccionada. En la Figura 27 se presentan algunos ejemplos de *Live Validation*.

The screenshot shows a form with several fields and their validation messages:

- Nombre:** An input field with the placeholder "Please enter your firstname".
- Genero:** A radio button group with "Masculino" selected. A message says "● This field is required".
- Los datos introducidos son correctos?** A dropdown menu with "■ Please select this field" highlighted.
- Clave:** An input field with the placeholder "Your password must be at least 8 characters long".
- Email:** An input field with the placeholder "Please enter a valid email address".
- Contraseña:** An input field.
- Confirmacion de contraseña:** An input field with the placeholder "Please enter the same password as above".

Figura 27 Ejemplos de *Live Validation*

4.2.1.5 RichAccordion

Este *widget* permite encapsular a varios elementos de interfaz de MoWebA dentro de paneles colapsables para presentar información en una cantidad limitada de espacio. Dentro de los elementos que pueden ser desplegados en los paneles, se encuentran los *UIElement* de cualquiera de sus extensiones *InputElement*, *OutputElement* o *ControlElement*, como así también los *CompositeUIElements*, *Table* y los *Form*. En la Figura 28 se presenta un ejemplo del *widget RichAccordion* que contiene cuatro paneles calapsables que contienen información sobre algunos lenguajes de programación para la web. El panel HTML 5 se encuentra seleccionado (activo) y por lo tanto es el panel que despliega la información, que para el ejemplo, se trata de texto HTML. Los otros tres paneles CSS 3, Javascript y Otros lenguajes se encuentran inactivos.

Programación Web

- » HTML 5
- » CSS 3
- » Javascript
- » Otros lenguajes

HTML5 (HyperText Markup Language, versión 5) es la quinta revisión importante del lenguaje básico de la World Wide Web. HTML. HTML5 especifica dos variantes de sintaxis para HTML: una «clásica», HTML (text/html), conocida como HTML5, y una variante XHTML conocida como sintaxis XHTML5 que deberá servirse con sintaxis XML (application/xhtml+xml).¹² Esta es la primera vez que HTML y XHTML se han desarrollado en paralelo. La versión definitiva de la quinta revisión del estándar se publicó en octubre de 2014. No se reconoció en viejas versiones de navegadores por sus nuevas etiquetas, se recomienda al usuario común actualizar su navegador a la versión más nueva, para poder disfrutar de todo el potencial que provee HTML5. El desarrollo de este lenguaje de marcado es regulado por el Consorcio W3C.

Figura 28 Ejemplo del widget RichAccordion

4.2.1.6 RichTabs

El *RichTabs* permite al igual que el *RichAccordion* agrupar a varios elementos de interfaz en cada una de sus pestañas o paneles como se lo denomina en MoWebA. En la Figura 29 se presenta el widget *RichTabs*, con un ejemplo que contiene tres pestañas, en la cual una de ellas se encuentra activa, presentando la correspondiente información, mientras las otras dos se encuentran inactivas.

Los tres máximos goleadores en los mundiales

- Miroslav Klose
- Ronaldo
- Gerd Mueller

Miroslav Klose

Miroslav Josef Klose (Opole, Polonia, 9 de junio de 1978) es un futbolista polaco naturalizado alemán, que se desempeña como delantero en la SS Lazio de la Serie A de Italia. Integro la selección alemana hasta la obtención de la Mundial de 2014 en Brasil, donde batió el récord de Ronaldo y se convirtió en el máximo goleador de la historia de la Copa Mundial de Fútbol. Klose es también el goleador histórico de la selección de fútbol de Alemania.

Figura 29 Ejemplo del widget RichTabs

4.2.2 El metamodelo de Estructura extendido

El metamodelo de *Estructura* no ha sufrido muchos cambios con respecto a su versión original. Dentro de las adaptaciones que se han tenido en cuenta en este metamodelo, se presentan los cambios llevados a cabo a los atributos de la clase *Properties*, *XPosition*, *YPosition*, *width* y *height*. Cada uno de estos atributos se divide en dos para distinguir su tipo y valor. Por lo tanto los atributos quedan como *XPositionType* y *XPositionValue*, *YPositionType* y *YPositionValue*, *widthType* y *widthValue* y finalmente *heightType* y *heightValue*. Los tipos de coordenadas, que forman parte de la enumeración *CoordType*, son pixel y percentage. Cualquiera de estas coordenadas puede

establecerse para configurar la posición de cada uno de los *CompositeUIElement* definidos en el metamodelo de *Contenido*.

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.

4.3 - EL PERFIL DE CONTENIDO Y ESTRUCTURA EXTENDIDO.

En la Figura 31 se muestra el perfil de *Contenido* para el modelado de los PIM de una aplicación con MoWebA. Al igual que el metamodelo de *Contenido*, el perfil de *Contenido* contiene las clasificaciones de elementos simples *UIElement* del tipo *InputElement*, *OutputElement* y *ControlElement*. Cada uno de los *UIElement* es representado por estereotipos que extienden de la metaclase *Property*, lo que implica que estos elementos serán representados como propiedades. Algunas de estas propiedades contienen valores etiquetados (*tagged values*) propios o heredados, que le permiten establecer ciertas características específicas a los estereotipos.

Los nuevos elementos stereotipados como *panels*, *richAccordion* y *richTabs* extienden a la metaclase *Package*, lo cual indica que estos elementos, serán representados por medio de *Packages*.

Con respecto al perfil de *Estructura*, el estereotipo *layout* que extiende a la metaclase *Package*, contempla todos los nuevos atributos agregados al metamodelo de *Estructura* presentados en la sección anterior. En la Figura 30 se presenta el perfil de *Estructura* extendido.

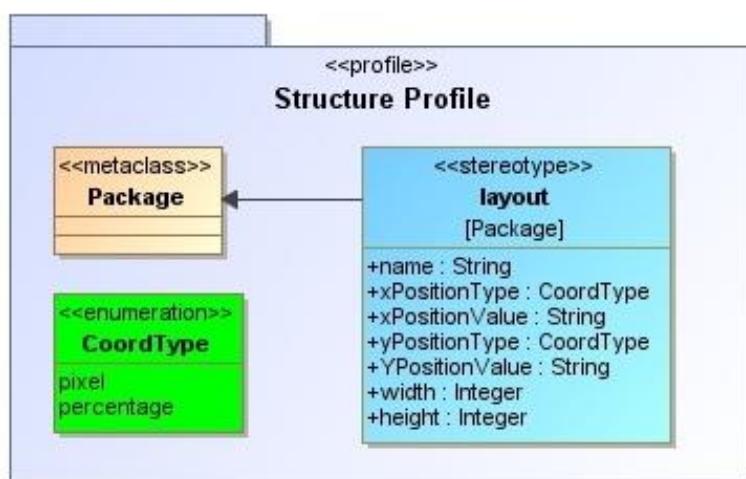


Figura 30 Perfil de Estructura extendido.

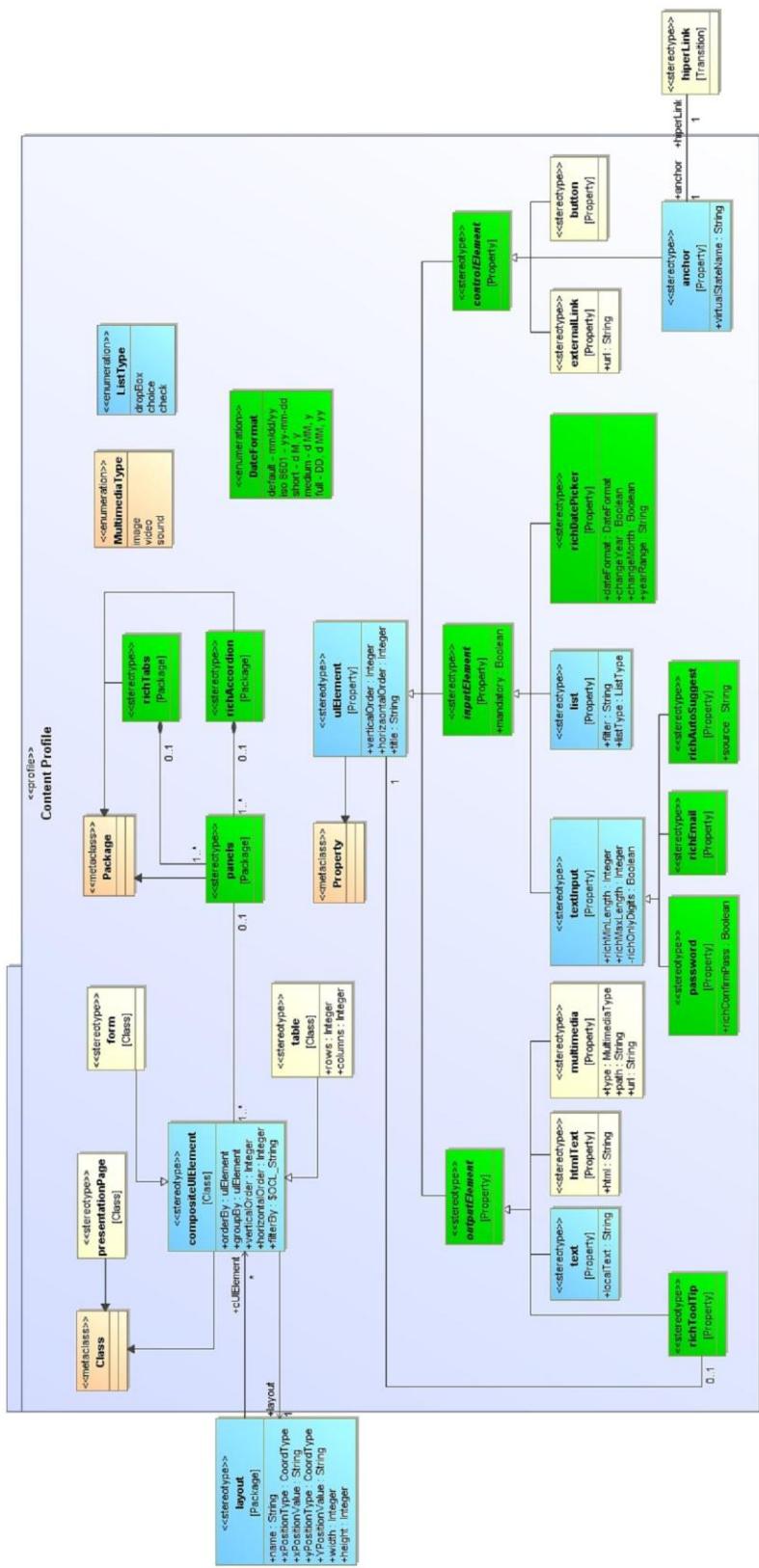


Figura 31 Perfil de contenido de MoWebA

4.4 MODELADO DE INTERFACES DE USUARIO CON MOWEBA.

En esta sección se presenta un ejemplo 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 32 se presenta el modelado del sistema *Person Manager* que se describe en el Anexo 1. Cada uno de los elementos que forman parte de la extensión a MoWebA se encuentran presentes en el PIM de Contenido se describen a continuación:

- a) El *richAccordion* que está compuesto de tres paneles que son: Agregar persona, Listar personas y Eliminar personas, representadas con el estereotipo *Panels*. Cada uno de los *Panels* contiene un elemento compuesto *compositeUIElement* que engloba a varios elementos simples *uiElements*. En los paneles Agregar persona y Eliminar persona se encuentran varios elementos simples que forman parte de la extensión RIA a MoWebA.
- b) Los campos *nombre* y *apellido* son campos con los estereotipos *textInput* y *richToolTip*. Ambos campos son obligatorios (*mandatory*) y despliegan mensajes que sugieren al usuario que se ingrese el nombre y el apellido completo.
- c) El campo fecha de nacimiento tiene el estereotipo *richDatePicker* que tiene los valores etiquetados *changeMonth* y *changeYear* activados. El formato de fecha elegido es el *default* (*mm-dd-yyyy*) y el rango de años desplegable (*yearRange*) se encuentra en el rango de 1970 al 2015.
- d) El campo país de origen está representado con el estereotipo *richAutoSuggest* y contiene la lista de sugerencias, en el archivo países.xml configurada en el valor etiquetado *source*.
- e) El campo género (con las opciones masculino, femenino definido en el valor etiquetado *filter*) del tipo *list*, es un campo de selección (*choice*) que debe seleccionarse obligatoriamente (*mandatory*).
- f) El campo email con el estereotipo *richEmail*, valida que el formato del campo sea del tipo email.
- g) El *textInput* usuario tiene en cuenta que la longitud del campo sea de al menos tres caracteres (*minLength=3*).
- h) El campo clave tiene establecido los estereotipos *password* y *richToolTip*. La clave es de ingreso obligatorio (*mandatory = true*) y debe reconfirmarse (*confirmPass=true*). El mensaje “Ingresé clave de al menos 8 caracteres” establecido en *title* será desplegado al posicionar el *mouse* sobre el campo clave.

- i) El campo confirmación de datos con el estereotipo *list* es un (check) que debe marcarse obligatoriamente para confirmar los datos ingresados.
- j) En el Formulario de eliminación, del panel Eliminar persona, se encuentra el *textInput* campo id de la persona, que es de ingreso obligatorio, permite solamente dígitos enteros positivos (*richOnlyDigits*) con un tamaño máximo de diez (*maxLength = 10*).

En la **Figura 32**, también puede apreciarse el diagrama de Estructura del *Person Manager*. En él se establecen las posiciones para cada uno de los elementos que forman parte de la aplicación. El diagrama de *Estructura* posee un paquete principal llamado *Layout* general que está compuesto de tres paquetes llamados *Cabecera*, *Estructura principal* y *Pié de página*. En cada uno de estos tres paquetes, se establecen las posiciones y tamaños de cada uno de los *compositeUIElement* que forman parte del diagrama de contenido. Los paquetes *Cabecera* y *Pié de página*, posicionan al título principal de la aplicación (*Person Manager*) y el texto que se establece en el *pié de página* (*Universidad Católica – 2015*). Los elementos que forman parte de los paquetes *Cabecera* y el *Pié de página*, se presentan en todas las vistas de la aplicación. El paquete *Estructura principal*, configura las posiciones en pixeles para cada uno de los *compositeUIElement* que forman del diagrama de contenido, que son *Nueva persona*, *Mostrar datos personales*, y el formulario de *eliminación*, respectivamente.

Finalmente en la Figura 33, se presenta la vista general de la interfaz de usuario final y los elementos enriquecidos que son representados en 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 la siguiente sección.

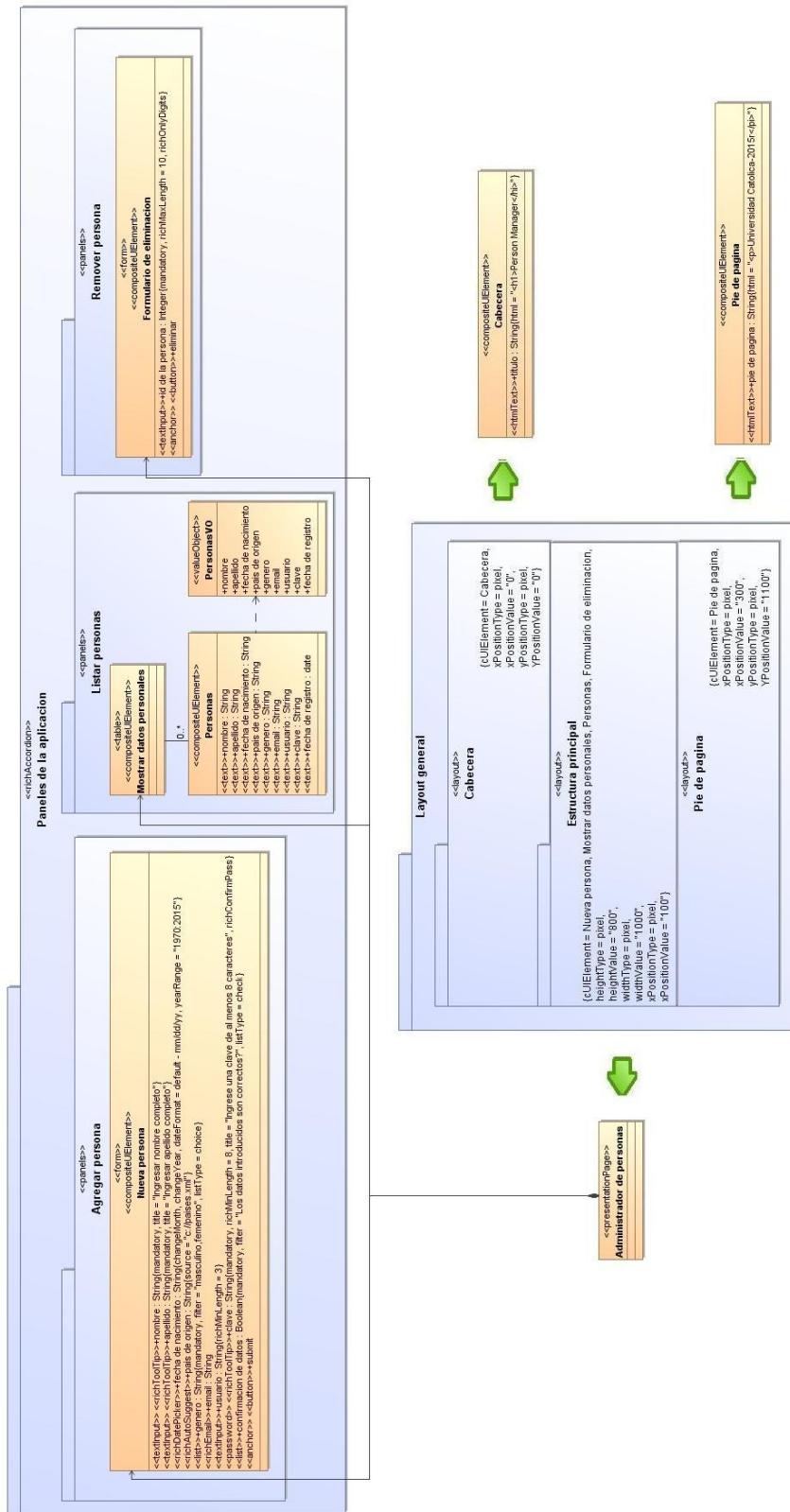


Figura 32 PIM del Person Manager con MoWebA

Figura 33 Interfaz final del *Person Manager* obtenida a partir del PIM.

4.5 LA TRANSFORMACIÓN DE MODELO A TEXTO (M2T) PARA LOS DIAGRAMAS DE CONTENIDO Y ESTRUCTURA DE MOWEBA

En esta sección se presentan 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. La herramienta M2T *Acceleo* es la herramienta utilizada para la generación de código.

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. En el Anexo 2 del libro se encuentran las plantillas completas de Presentación y Estructura.

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

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 código 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 qué elementos del modelo y qué 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.

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. El código fuente 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.

4.6 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

4.6.1 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 y 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 si 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*. 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.

La plantilla de *Estructura* transforma las posiciones definidas en pixeles o en porcentajes, 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.

4.6.2 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 (RIA) como los *richToolTip*, *richAccordion*, *richTabs*, *richDatePicker*, *richAutoSuggest* y los *field live validation*, 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 RIA, debido a que permiten machejar 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 34, se presenta un ejemplo del proceso de transformación para el elemento *richDatePicker*. En primera instancia 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 *default -mm-dd-yy* y que el rango 1970:2015 será desplegado en una lista.

```

<<form>>
<<compositeUIElement>>
    Nueva persona
<<textInput>> <<richToolTip>>+nombre : String(mandatory, title = "Ingresar nombre completo")
<<textInput>> <<richToolTip>>+apellido : String(mandatory, title = "Ingresar apellido completo")
<<richDatePicker>>+fecha de nacimiento : String(changeMonth, changeYear, dateFormat = default - mm/dd/yy, yearRange = "1970:2015")
<<richAutoSuggest>>+pais de origen : String(source = "c:/países.xml")
<<list>>+genero : String(mandatory, filter = "masculino,femenino", listType = choice)
<<richEmail>>+email : String
<<textInput>>+usuario : String(richMinLength = 3)
<<password>> <<richToolTip>>+clave : String(mandatory, richMinLength = 8, title = "Ingrese una clave de al menos 8 caracteres", richConfirmPass)
<<list>>+confirmacion de datos : Boolean(mandatory, filter = "Los datos introducidos son correctos?", listType = check)
<<anchor>> <<button>>+submit

```

```

</script>
    [for (c2: Class | c.followingSiblings(Class))]
        [for (att: Property | c2.getAllAttributes()->asSequence())]
            [for (s: Stereotype | att.getAppliedStereotypes()->asSequence())]
                [elseif s.name.toString() = 'richDatePicker']
                    $function() {
                        $( "#[att.name.substituteAll(' ', '') /]" ).datepicker({
                            [if att.getValue(s,'changeYear') = true]
                                changeYear: true,
                                [/if]
                                [if att.getValue(s,'changeMonth') = true]
                                    changeMonth: true,
                                    [/if]
                                    [if att.getValue(s, 'yearRange').toString().size() <> 0]
                                        yearRange: "[att.getValue(s, 'yearRange').toString() /]",
                                        [/if]
                                        [if att.getValue(s, 'dateFormat').toString().strstr('Default - mm/dd/yy') = true ]
                                            dateFormat: "mm/dd/yy"
                                            [elseif att.getValue(s, 'dateFormat').toString().strstr('ISO 8601 - yy-mm-dd') = true ]
                                                dateFormat: "yy-mm-dd"
                                                [elseif att.getValue(s, 'dateFormat').toString().strstr('Short - d M, y') = true ]
                                                    dateFormat: "d M, y"
                                                    [elseif att.getValue(s, 'dateFormat').toString().strstr('Medium - d MM, y') = true ]
                                                        dateFormat: "d MM, y"
                                                        [elseif att.getValue(s, 'dateFormat').toString().strstr('Full - DD, d MM, yy') = true ]
                                                            dateFormat: "DD, d MM, yy"
                                                            [/if]
                                                        });
                                                    });
                                                };
                                            
```

```

[comment generaciÃ³n del body para el html /]
<body>
    [for (c2: Class | c.followingSiblings(Class))]
        [for (att: Property | c2.getAllAttributes()->asSequence())]
            [for (s: Stereotype | att.getAppliedStereotypes()->asSequence())]
                [if s.name.toString() = 'richDatePicker']
                    <p>@[att.name.toUpperCaseFirst() /] <input type="text" id="@[att.name.substituteAll(' ', '') /]" name="@[att.name.substituteAll(' ', '') /]" [if att.getValue(s, 'title').size() <> 0] title="@[att.getValue(s, 'title').toString() /]"[/if]</p>
                    [if att.getValue(s, 'title').size() <> 0] title="@[att.getValue(s, 'title').toString() /]"[/if]</p>
                [/for]
            [/for]
        
```

Figura 34 PIM de contenido de MoWebA y templates de trasnformación de Contenido para el richDatePicker

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 35 se observa el código HTML generado por la plantilla de transformación para el elemento *richDatePicker* definido en el ejemplo de la Figura 34.

```
<script>
$(function() {
    $("#fechadenacimiento").datepicker({
        changeYear: true,
        changeMonth: true,
        yearRange: "1960:2015",
        dateFormat: "yy-mm-dd"
    });
});
</script>

<body>
    <p>Fecha de nacimiento <input type="text" id="fechadenacimiento" name="fechadenacimiento">
</body>
```

Figura 35 Código fuente HTML generado para el richDatePicker generado a partir de las plantillas

4.6.3 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 un paquete 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 36 se presenta el *template* de transformación de Estructura.

```

2 [module generate('http://www.eclipse.org/uml2/4.0.0/UML')]
3 [import org::eclipse::acceleo::module::layout::services::generate /]
4@[template public generateElement(p : Package)]
5   [comment @main /]
6   [if p.hasStereotype('layout')]
7     [file (p.name.concat('.css'), false, 'UTF-8')]
8 //[[protected (p.name)]
9 body {background-color: #3f7506 }
10 //[[ /protected]
11   [for (s: Stereotype | p.getAppliedStereotypes()->asSequence())]
12     [for (c: Class | p.getValue(s,'cUIElement')) separator('\n')]
13 #[c.name.substituteAll(' ', '')] {
14   position: relative;
15     [if p.getValue(s,'height').toString().toInteger() <> 0]
16   height: [p.getValue(s,'height')+'px'+';']
17   [/if]
18     [if p.getValue(s,'width').toString().toInteger() <> 0]
19   width: [p.getValue(s,'width')+'px'+';']
20   [/if]
21     [if p.getValue(s,'xPosition').toString().toInteger() <> 0]
22   left: [p.getValue(s,'xPosition')+'px'+';']
23   [/if]
24     [if p.getValue(s,'yPosition').toString().toInteger() <> 0]
25   top: [p.getValue(s,'yPosition')+'px'+';']
26   [/if]
27 }
28   [/for]
29   [/for]
30   [/file]
31 [/if]
32 [/template]

```

Figura 36 Plantilla de transformación para el posicionamiento de elementos

4.7 RESUMEN DEL CAPITULO

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.

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 *Estructura*, en donde se mostró como se establecen las coordenadas de cada uno de los elementos. Seguidamente se ilustró un ejemplo de PIM con las extensiones RIA propuestas a MoWebA, junto a algunas vistas de tomas de pantalla de la aplicación.

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

5.1 INTRODUCCIÓN

En este capítulo se describirá el proceso llevado a cabo para realizar una validación preliminar de MoWebA con extensiones RIA. La validación consiste en comparar la capa de presentación de MoWebA con extensiones RIA con respecto al mismo enfoque pero 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*.

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 [2].

Según Runeson [27], 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.

Atendiendo a la anterior definición, no siempre es posible realizar un caso de estudio. Una alternativa es la ilustración, que a pesar de no ser un método de validación formal, sirve para presentar a una audiencia cómo funciona un artefacto y cómo este puede resolver un *toy problem* en particular, permitiendo llegar a una conclusión intuitiva [28].

En este capítulo se utiliza una ilustración para validar preliminarmente las extensiones RIA de MoWebA por medio de la resolución de un *toy problem* denominado *Person Manager*. Para brindar mayor formalidad a la ilustración, la misma se realizó siguiendo las guías propuestas por Runeson en la definición y análisis de casos de estudio.

5.2 DISEÑO DE LA ILUSTRACIÓN

5.2.1 Objetivos

Esta ilustración se presenta con la intención de ofrecer un análisis crítico de las extensiones RIA llevadas a cabo con el enfoque MoWebA desde el punto de vista de las interfaces enriquecidas. La 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.

Esta ilustración busca obtener datos lo suficientemente reveladores que permitan intuir que la propuesta de extensión a nivel de la capa de presentación para el lado del cliente llevada a cabo al enfoque MoWebA, ofrece cobertura a algunas de las diversas características que contemplan las RIA 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 contemplan 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*. El *Person Manager* es una aplicación Web que contiene en sus especificaciones funcionales características de las RIA y resulta lo suficientemente expresiva para ilustrar la propuesta de extensión.

Analizar la productividad en el proceso de modelado de los PIM en MoWebA con RIA, también es uno de los objetivos de esta ilustración, en la cual se intenta comparar los tiempos de modelado para ambos enfoques, como así también las veces que es necesario generar el código fuente de la aplicación a partir de la corrección de los PIM, hasta obtener la interfaz final. Finalmente se busca comparar la cantidad de líneas de código generadas de manera automática en ambos enfoques.

5.2.1.1 Objetivos específicos

Comparar a MoWebA sin RIA y a MoWebA con extensiones RIA, con respecto al tiempo de modelado y a la cantidad de generaciones y refinamientos a nivel de modelado que se deben hacer hasta obtener una interfaz final satisfactoria.

Verificar si MoWebA con extensiones RIA, ofrece ventajas sobre MoWebA sin RIA con respecto a las presentaciones enriquecidas y con respecto a la lógica en el lado del cliente.

Analizar la cantidad de líneas de código generadas de manera automática a partir de los modelos PIM, para ambos enfoques.

5.2.2 Preguntas de investigación

A partir de los objetivos anteriormente citados, surgen las siguientes preguntas de investigación para esta ilustración:

PI1: ¿Consumo una mayor cantidad de tiempo modelar la aplicación aplicando MoWebA con RIA que MoWebA sin RIA?

PI2: ¿Para cuál de los enfoques es necesaria una mayor cantidad de generaciones de código para obtener la interfaz de usuario final?

PI3: Desde el punto de vista de las presentaciones enriquecidas, ¿qué ventajas aportan las características RIA presentes en la aplicación implementada con MoWebA con RIA con respecto a MoWebA sin RIA?

PI4: Desde el punto de vista de la lógica de negocios en el lado del cliente, ¿qué ventajas aportan las características RIA presentes en la aplicación implementada con MoWebA con RIA con respecto a MoWebA sin RIA?

PI5: Para cada una de las vistas del *Person Manager*, ¿qué cantidad de líneas de código para la interfaz de usuario se pudieron generar de manera automática a partir de los modelos, en cada uno de los enfoques implementados?

5.2.3 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.

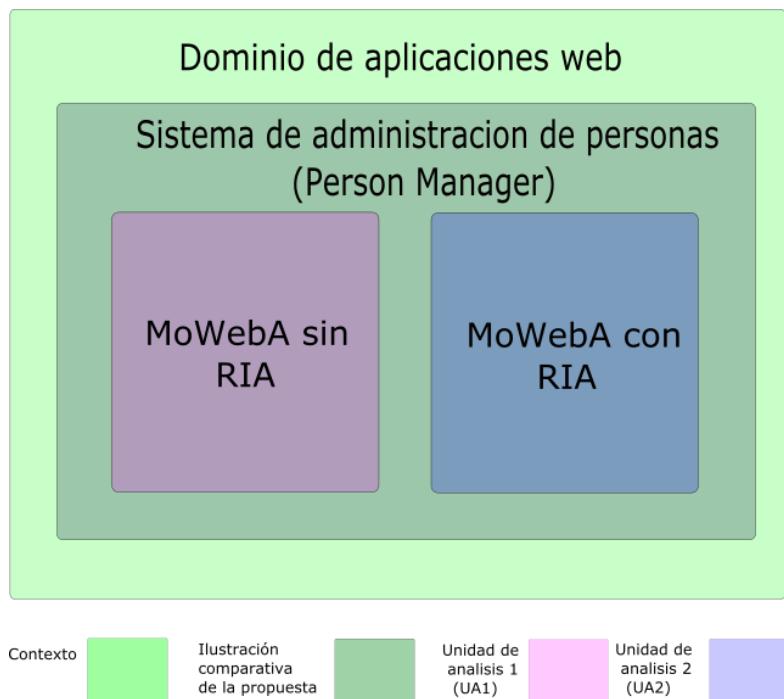


Figura 37 Ilustración del sistema *Person Manager* implementado con MoWebA desde dos enfoques distintos

El caso fue analizado desde dos unidades de análisis. La primera unidad de análisis se refiere a la implementación de la capa de presentación del *Person Manager* con MoWebA sin RIA. La segunda unidad de análisis se refiere a la implementación de la misma capa de presentación del caso estudiado, con la nueva propuesta de extensión RIA a MoWebA. Por lo tanto, se trata de una ilustración embebida, ya que se cuenta con más de una unidad de análisis para un mismo caso, como se puede apreciar en la Figura 37.

En el Anexo 1 se presenta la descripción del sistema *Person Manager* de una manera general, es decir, se describen los requerimientos funcionales básicos y se agregan algunos requerimientos adicionales, que son precisamente, requerimientos RIA. Las unidades de análisis representan a los dos enfoques implementados con MoWebA (con o sin RIA) que servirán para obtener respuestas a las preguntas de investigación. El *Person Manager* está basado en el trabajo de Gharavi [31].

5.2.4 Variables

Las variables de medición necesarias para responder las preguntas de investigación PI1, PI2 y PI5 se definen a continuación:

$i = \{1, 2, 3\}$, en donde **1** corresponde a la vista *Agregar Persona*, **2** corresponde a la vista *Listar Personas* y **3** corresponde a la vista *Eliminar Persona*.

$j = \{a, b\}$, en donde **a** es el enfoque MoWebA sin RIA y **b** es el enfoque MoWebA con RIA.

$t_{i,j}$ = tiempo total en minutos, empleado para el modelado de la vista i empleando el enfoque j .

$n_{i,j}$ = número total de generaciones de código para la vista i utilizando el enfoque j .

$T_a = t_{i=1,a} + t_{i=2,a} + t_{i=3,a}$, es el tiempo total en minutos, empleado para modelar todas las vistas i para $j = a$ o MoWebA sin RIA.

$T_b = t_{i=1,b} + t_{i=2,b} + t_{i=3,b}$, es el tiempo total empleado para modelar todas las vistas i para $j = b$ o MoWebA con RIA.

$N_a = n_{i=1,a} + n_{i=2,a} + n_{i=3,a}$, es el número total de generaciones de código necesarias para obtener la interfaz usuario final de todas las vistas i para $j = a$ o MoWebA sin RIA.

$N_b = n_{i=1,b} + n_{i=2,b} + n_{i=3,b}$, es el número total de generaciones de código necesarias para obtener la interfaz usuario final de todas las vistas i para $j = b$ o MoWebA con RIA.

$Ica_{i,j}$ = total de líneas de código generadas de manera automática a partir de los modelos PIM de entrada para la vista i empleando el enfoque j .

$Icm_{i,j}$ = total de líneas implementadas manualmente para la vista i empleando el enfoque j .

$LCA_a = Ica_{i=1,a} + Ica_{i=2,a} + Ica_{i=3,a}$, total de líneas de código generadas de manera automática a partir de los modelos PIM de entrada empleando el enfoque a.

$LCM_a = Icm_{i=1,a} + Icm_{i=2,a} + Icm_{i=3,a}$, total de líneas de código implementadas manualmente empleando el enfoque a.

$LCA_b = Ica_{i=1,b} + Ica_{i=2,b} + Ica_{i=3,b}$, total de líneas de código generadas de manera automática a partir de los modelos PIM de entrada empleando el enfoque b.

$LCM_b = lcm_{i=1,b} + lcm_{i=2,b} + lcm_{i=3,b}$, total de líneas de código implementadas manualmente empleando el enfoque b.

Las preguntas de investigación 3 y 4 no necesitan de la definición de variables para obtener sus respuestas, sino más bien, por medio de capturas de pantalla comparativas y descripciones detalladas de podrá dar respuesta a las mismas.

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 de las generaciones, el código fuente se va refinando a partir de la modificación de los modelos PIM de entrada.

5.2.5 Métodos de colección de datos

Esta ilustración se realizó utilizando a un individuo como población (el autor de la tesis), tanto para la implementación de las diferentes unidades de análisis, como para la recolección de los datos necesarios para responder PI1, PI2 y PI5, por lo tanto no se llevaron a cabo entrevistas ni encuestas que son empleadas de manera común en casos de estudio y experimentos.

Teniendo en cuenta este hecho particular, los datos necesarios para calcular las variables de respuesta, fueron recolectados a medida que las unidades funcionales de la aplicación (las vistas) del *Person Manager* eran implementadas. Primeramente se recolectó toda la información correspondiente a MoWebA sin RIA y luego se procedió a la colección de los datos de MoWebA con RIA. Para cada uno de los enfoques, primeramente se recolectaron todos los datos correspondientes a una vista en particular hasta la conclusión de esta. Seguidamente se pasaba a la siguiente vista y se recababan los datos correspondientes y así sucesivamente.

Para la recolección de los tiempos de modelado para cada una de las vistas del *Person Manager* en los distintos enfoques, se utilizó la 7, en donde se presentan las variables que deben ser colectadas.

Tiempos en minutos de modelado y generación de código para cada i,j	MoWebA sin RIA (j = a)	MoWebA con RIA (j = b)
Agregar persona (i = 1)	$t_{i=1,a}$	$t_{i=1,b}$
Listar personas (i = 2)	$t_{i=2,a}$	$t_{i=2,b}$
Remover persona(i = 3)	$t_{i=3,a}$	$t_{i=3,b}$
TOTALES	$T_{ew}{}_{j+a} = t_{i=1,a} + t_{i=2,a} + t_{i=3,a}$	$T_b = t_{i=1,b} + t_{i=2,b} + t_{i=3,b}$

Tabla 7 Tiempo de modelado para cada uno de las vistas del Person Manager en cada uno de los enfoques.

Para recolectar el número de generaciones de código para cada una de las vistas del *Person Manager* en los distintos enfoques, se utilizó la Tabla 8, en donde se presentan las variables que deben ser colectadas.

Número de generaciones de código para cada i,j	MoWebA sin RIA ($j = a$)	MoWebA con RIA ($j = b$)
Agregar persona ($i = 1$)	$n_{i=1,a}$	$n_{i=1,b}$
Listar personas ($i = 2$)	$n_{i=2,a}$	$n_{i=2,b}$
Remover persona ($i = 3$)	$n_{i=3,a}$	$n_{i=3,b}$
Totales	$N_a = n_{i=1,a} + n_{i=2,a} + n_{i=3,a}$	$N_b = n_{i=1,b} + n_{i=2,b} + n_{i=3,b}$

Tabla 8 Número de generaciones de código para cada una de las vistas del *Person Manager* en cada uno de los enfoques

Finalmente, para llevar a cabo un análisis de las líneas de código generadas automáticamente a partir de los modelos de entrada se utilizó la Tabla 9

Líneas de código	MoWebA sin RIA ($j = a$)			MoWebA con RIA ($j = b$)		
	Líneas de código automáticas	Líneas de código manuales	Totales	Líneas de código automáticas	Líneas de código manuales	Totales
Agregar persona ($i = 1$)	$lca_{i=1,a}$	$lcm_{i=1,a}$	$lca_{i=1,a} + lcm_{i=1,a}$	$lca_{i=1,b}$	$lcm_{i=1,b}$	$lca_{i=1,b} + lcm_{i=1,b}$
Listar personas ($i = 2$)	$lca_{i=2,a}$	$lcm_{i=2,a}$	$lca_{i=2,a} + lcm_{i=2,a}$	$lca_{i=2,b}$	$lca_{i=2,b}$	$lca_{i=2,b} + lca_{i=2,b}$
Remover persona ($i = 3$)	$lca_{i=3,a}$	$lcm_{i=3,a}$	$lca_{i=3,a} + lcm_{i=3,a}$	$lca_{i=3,b}$	$lcm_{i=3,b}$	$lca_{i=3,b} + lcm_{i=3,b}$
Totales	$LCA_a = lca_{i=1,a} + lca_{i=2,a} + lca_{i=3,a}$	$LCM_a = lcm_{i=1,a} + lcm_{i=2,a} + lcm_{i=3,a}$	$LCA_a + LCM_a =$	$LCA_b = lca_{i=1,b} + lca_{i=2,b} + lca_{i=3,b}$	$LCM_b = lcm_{i=1,b} + lcm_{i=2,b} + lcm_{i=3,b}$	$LCA_b + LCM$

Tabla 9 Tabla utilizada para la comparación de líneas de código

5.2.6 Métodos de análisis de los datos

Para cada una de las variables definidas en la sección 5.2.5, sólo se va a tener una medición por cada variable y para cada uno de los enfoques, y por lo tanto no se va a poder realizar un análisis estadístico formal, sino más bien, se van a comparar los valores obtenidos en cada una de las implementaciones del *Person Manager*. Estos valores recolectados servirán para reportar las respuestas a cada unas de las preguntas de investigación.

5.2.7 Amenazas a la validez de los datos

Como se mencionó anteriormente, se llevó a cabo una ilustración comparativa, en la cual se optó por un *toy problem* en el contexto de las aplicaciones Web, el cual fue implementado por el autor del trabajo. Se optó por el *Person Manager*, ya que es posible representar en él las nuevas características extendidas a MoWebA de manera clara y concisa. Para que los datos de las mediciones obtenidas sean lo más fidedignas posibles, se intentó mitigar las amenazas a la validez de los datos llevando ciertas acciones preventivas.

Con respecto a las medidas tomadas en esta ilustración para minimizar los factores de riesgo en la validez de los datos obtenidos, se ha tenido en consideración los siguientes puntos:

- a) Primeramente el autor del trabajo recibió entrenamiento sobre la metodología MoWebA en su original (sin RIA) y trabajó con diversos ejemplos distintos al de esta ilustración para modelar y generar aplicaciones. Seguidamente extendió el metamodelo original junto a los perfiles e implementó el nuevo generador de código. Entonces no hizo falta entrenamiento previo para el modelado de los PIM en ambos enfoques del *Person Manager*. Previamente se llevó a cabo un *testing* exhaustivo de la herramienta de generación de código para evitar problemas.
- b) Se empleó completamente el enfoque MoWebA sin RIA hasta obtener la interfaz final del *Person Manager* luego el enfoque MoWebA con RIA de igual manera con una semana de diferencia entre la implementación de cada enfoque. Se tomó esta decisión, ya que si se implementa una vista con el enfoque MoWebA sin RIA y luego la misma vista con el enfoque MoWebA con RIA, se puede obtener demasiada familiaridad con el modelado de la última de las vistas implementada, y por ende no sería tan realista la medición de los tiempos de modelado en el enfoque MoWebA con RIA, ya que estos podrían reducirse.
- c) A medida que se iba implementando cada una de las vistas en un enfoque en particular, se iban recabando los datos correspondientes para las variables de medición para esa vista en particular. Se pasaba a implementar la siguiente vista de la aplicación, una vez culminada en su totalidad la vista actual con todos los datos analíticos recabados. Se tomó esta decisión para hacer una medición más precisa de los tiempos y evitar estimaciones arbitrarias.
- d) Para la medición de los tiempos de modelado de los PIM se designó a una persona ajena al proyecto que tomaba los valores correspondientes desde el inicio del PIM hasta su fin para una mayor objetividad en las mediciones.
- e) Para una mayor calidad en el análisis de líneas de código se optó por el uso de la herramienta de uso libre CLOC³⁶. Con esta herramienta se evitó el tener que llevar a cabo el conteo de líneas de código manualmente.

En cierto sentido las medidas tomadas sirven para mitigar algunas amenazas posibles a la validez de los datos recabados, pero no se pueden eliminar todas las posibles. Teniendo en cuenta que el autor del trabajo 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 datos como es esperado en casos de estudio o experimentos. Es bajo esta circunstancia, que se decidió llevar a cabo una ilustración de la propuesta de extensión y no

³⁶ CLOC: <http://cloc.sourceforge.net/> 2015

un caso de estudio, ya que los resultados y conclusiones obtenidas, deben ser considerados en el contexto en el que fueron recabados.

5.2.8 Colección de los datos

En esta sección se presentan los datos recabados para responder a las 5 preguntas de investigación. La información asociada a las PI1, PI2 y PI5 está asociadas a la Tabla 10, Tabla 11 y Tabla 12 y respectivamente que fueron definidas en la sección

Tiempos en minutos de modelado y generación de código para cada i,j	MoWebA sin RIA (j = a)	MoWebA con RIA (j = b)
Agregar persona (i = 1)	50 minutos	56 minutos
Listar persona (i = 2)	27 minutos	28 minutos
Remover persona(i = 3)	29 minutos	30 minutos
Totales	106 minutos	114 minutos

Tabla 10 Tiempo de modelado para cada uno de los enfoques

Número de generaciones de código para cada i,j	MoWebA sin RIA (j = a)	MoWebA con RIA (j = b)
Agregar persona (i = 1)	3	4
Listar persona (i = 2)	1	1
Remover persona(i = 3)	2	3
Totales	6	8

Tabla 11 Cantidad de generaciones de código para cada uno de los enfoques para la obtención de la interfaz final

Líneas de código	MoWebA sin RIA (j = a)			MoWebA con RIA (j = b)		
	Líneas de código automáticas	Líneas de código manuales	Total es	Líneas de código automáticas	Líneas de código manuales	Totales
Agregar persona (i = 1)	51	56	107	135	56	191
Mostrar persona (i = 2)	1	45	46	3	45	48
Remover persona(7	27	34	31	27	58

i = 3)						
Estructura y código común para todas las vistas(cabecera, estructura y pie de pagina	67	10	77	52	38	90
Totales	126	138	264	221	166	387

Tabla 12 Líneas de código para ambos enfoques del *Person Manager*

The figure consists of two side-by-side screenshots of a web application interface. Both screenshots show a header bar with tabs for 'Home', 'localhost:21353', and 'localhost:4054'. The header also includes search and filter icons.

Left Screenshot (Without RIA):

- Header:** Shows 'Aplicaciones' and 'Otros marcadores'.
- Title:** 'Person Manager'.
- Buttons:** 'Agregar persona', 'Listar personas', and 'Remover persona'.
- Form Fields:**
 - Nombre:
 - Apellido:
 - Fecha de nacimiento:
 - País de origen:
 - Sexo:
 - Masculino
 - Femenino
 - Email:
 - Usuario:
 - Clave:
 - Confirmar clave:
- Submit Button:** A green 'Submit' button.
- Page Footer:** 'Universidad Católica - 2015'

Right Screenshot (With RIA):

- Header:** Shows 'Aplicaciones' and 'Otros marcadores'.
- Title:** 'Person Manager'.
- Buttons:** 'Agregar personas', 'Listar personas', and 'Remover persona'.
- Form Fields:**
 - Nombre:
 - Apellido:
 - Fecha de nacimiento:
 - País de origen:
 - Sexo:
 - Masculino
 - Femenino
 - Email:
 - Usuario:
 - Clave:
 - Confirmar clave:
- Message:** 'Los datos introducidos son correctos?' with a checkmark icon.
- Submit Button:** A green 'Submit' button.
- Page Footer:** 'Universidad Católica - 2015'

Figura 38 MoWebA con y sin RIA para la vista Agregar Persona

Person Manager

Id	Nombre	Apellido	Fecha de nacimiento	País	Sexo	Email	Turno	Contraseña	Fecha de registro
8	Mickey	Mouse	03/14/2015	AdoScript	male	jcarlos@ahoo.com	jcarlos	jcarlos@ahoo.com	19-02-20
9	Mickey	Mouse	03/26/2015	Cajira	male	jimic@terri.com.br	jimic	comor	19-03-17
10	Mickey	Mouse	03/18/2015	Java	female	wedo.gr@gmail.com	wedo	wedo@gmail.com	19-04-12
11	Kimy	Toulouse	03/10/2015	CodeFusion	female	lmy@actvision.com	lmy	krison	20-11-00
12	Kimy	Toulouse	03/10/2015	CodeFusion	female	lmy@actvision.com	lmy	krison	20-11-06
13	Mark	Twain	03/20/2015	JavaScript	male	hol@gmail.com	contare	aswdfq	20-12-49
14	Jack	Johnson	03/06/2015	COBOL	male	jack.johnson@google.com	jack.johnson	lifejacket	20-13-39
23	Ivan	Lopez	13/04/1981	Paraguay	masculino	ivelopez@gmail.com	ivelopez	ivelopez@gmail.com	15-06-09
24	Pablo	Chamorro	03/13/2015	Paraguay	masculino	pablop@personal.com.py	pablop	pablop@personal.com.py	18-11-47
25	Gustavo	Gonzalez	18/09/1980	Paraguay	masculino	gonzalug@personal.com.py	gonzalug	gonzalug@personal.com.py	18-09-12
26	Juan	Gomez	11/12/1970	Argentina	masculino	jung@homed.com	jung	jung@hotmail.com	18-7-38
27									20-12-37
28									20-12-37
29									20-12-37
30									20-12-37
31									20-12-37
32									20-12-37
33									20-12-37
34									20-12-37
35									20-12-37
36									20-12-37
37									20-12-37
38									20-12-37
39									20-12-37
40									20-12-37
41									20-12-37
42									20-12-37
43									20-12-37
44									20-12-37
45									20-12-37
46									20-12-37
47									20-12-37
48									20-12-37
49									20-12-37
50									20-12-37
51									20-12-37
52									20-12-37
53									20-12-37
54									20-12-37
55									20-12-37
56									20-12-37
57									20-12-37
58									20-12-37
59									20-12-37
60									20-12-37
61									20-12-37
62									20-12-37
63									20-12-37
64									20-12-37
65									20-12-37
66									20-12-37
67									20-12-37
68									20-12-37
69									20-12-37
70									20-12-37
71									20-12-37
72									20-12-37
73									20-12-37
74									20-12-37
75									20-12-37
76									20-12-37
77									20-12-37
78									20-12-37
79									20-12-37
80									20-12-37
81									20-12-37
82									20-12-37
83									20-12-37
84									20-12-37
85									20-12-37
86									20-12-37
87									20-12-37
88									20-12-37
89									20-12-37
90									20-12-37
91									20-12-37
92									20-12-37
93									20-12-37
94									20-12-37
95									20-12-37
96									20-12-37
97									20-12-37
98									20-12-37
99									20-12-37
100									20-12-37
101									20-12-37
102									20-12-37
103									20-12-37
104									20-12-37
105									20-12-37
106									20-12-37
107									20-12-37
108									20-12-37
109									20-12-37
110									20-12-37
111									20-12-37
112									20-12-37
113									20-12-37
114									20-12-37
115									20-12-37
116									20-12-37
117									20-12-37
118									20-12-37
119									20-12-37
120									20-12-37
121									20-12-37
122									20-12-37
123									20-12-37
124									20-12-37
125									20-12-37
126									20-12-37
127									20-12-37
128									20-12-37
129									20-12-37
130									20-12-37
131									20-12-37
132									20-12-37
133									20-12-37
134									20-12-37
135									20-12-37
136									20-12-37
137									20-12-37
138									20-12-37
139									20-12-37
140									20-12-37
141									20-12-37
142									20-12-37
143									20-12-37
144									20-12-37
145									20-12-37
146									20-12-37
147									20-12-37
148									20-12-37
149									20-12-37
150									20-12-37
151									20-12-37
152									20-12-37
153									20-12-37
154									20-12-37
155									20-12-37
156									20-12-37
157									20-12-37
158									20-12-37
159									20-12-37
160									20-12-37
161									20-12-37
162									20-12-37
163									20-12-37
164									20-12-37
165									20-12-37
166									20-12-37
167									20-12-37
168									20-12-37
169									20-12-37
170									20-12-37
171									20-12-37
172									20-12-37
173									20-12-37
174									20-12-37
175									20-12-37
176									20-12-37
177									20-12-37
178									20-12-37
179									20-12-37
180									20-12-37
181									20-12-37
182									20-12-37
183									20-12-37
184									20-12-37
185									20-12-37
186									20-12-37
187									20-12-37
188									20-12-37
189									20-12-37
190									20-12-37
191									20-12-37
192									20-12-37
193									20-12-37
194									20-12-37
195									20-12-37
196									20-12-37
197									20-12-37
198									20-12-37
199									20-12-37
200									20-12-37
201									20-12-37
202									20-12-37
203									20-12-37
204									20-12-37
205									20-12-37
206									

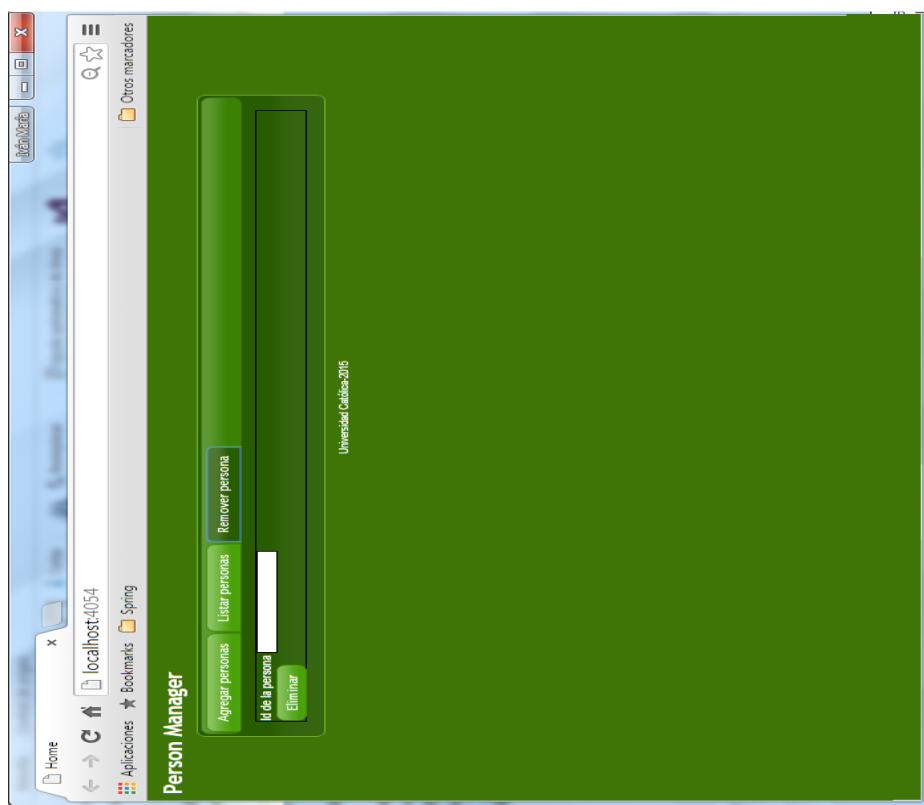
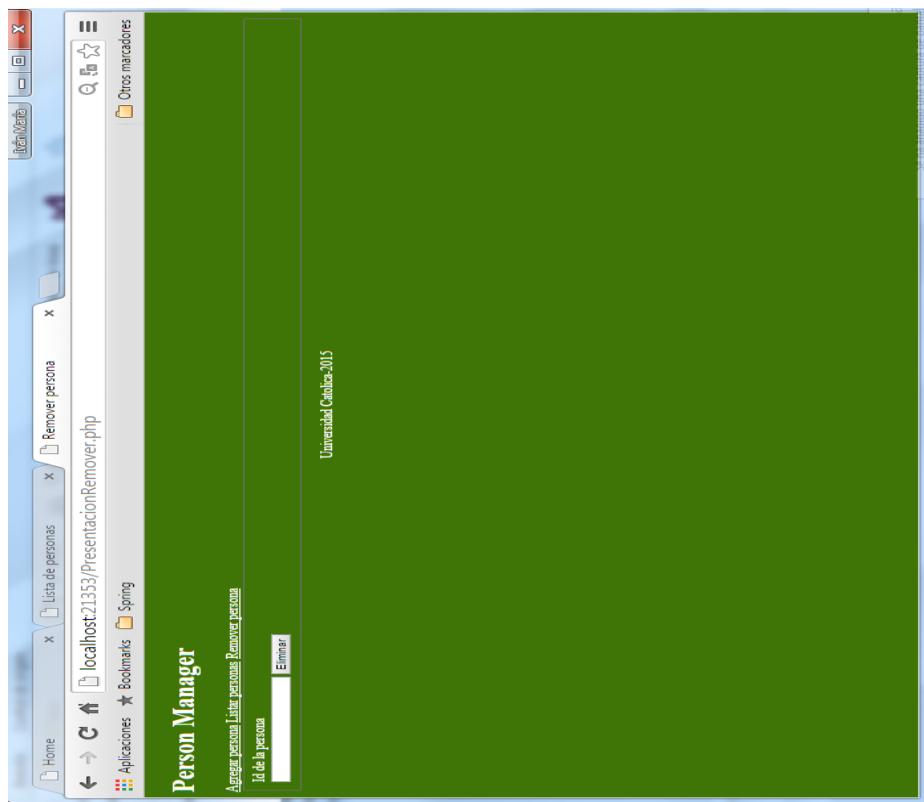


Figura 40 MoWebA con y sin RIA para la vista Remover persona

5.2.9 Análisis e interpretación de los resultados.

Aquí se presentan los resultados de las mediciones realizadas en la sección anterior, En base a los datos obtenidos, se responden las preguntas de investigación.

PI1: ¿Consume una mayor cantidad de tiempo modelar la aplicación aplicando MoWebA con RIA que MoWebA sin RIA?

Según puede apreciarse en la Tabla 7, a nivel general existe una diferencia de 8 minutos entre los enfoques aplicados al *Person Manager*. El enfoque MoWebA con RIA conlleva a definir en los modelos PIM, diversas características (expresadas por medio de valores etiquetados) particulares para cada uno de los elementos RIA presentes. Para el enfoque MoWebA sin RIA es no ocurre de igual forma, ya que son muchas menos las propiedades que pueden definirse sobre cada uno de los elementos que forman parte del perfil de Contenido. Por lo tanto al tener que definir más valores etiquetados para el caso MoWebA con RIA se incurre en una mayor cantidad de tiempo en el modelado.

Particularmente para la vista *Agregar Persona* que corresponde a la vista con mayor número de requerimientos RIA, existe la mayor diferencia de tiempo en el modelado (6 minutos más en el enfoque con RIA que su contraparte sin RIA) y resulta natural este hecho debido a la cantidad de valores etiquetados que hay que establecer en casi todos los elementos del formulario. En esta vista tenemos *richDatePicker*, *richAutoSuggest*, *richToolTips* y validaciones de diversos campos.

Para la vista *Listar Personas* no hubo mucha diferencia en el tiempo de modelado (tan solo 1 minuto más en el enfoque con RIA), lo que no resulta muy relevante, ya que el PIM de esta vista es la misma para ambos enfoques al no tener características RIA.

Finalmente la vista *Remover persona* en el enfoque MoWebA con RIA llevó 1 minuto más de tiempo de modelado que MoWebA sin RIA. Aquí la diferencia es mínima y esto se debe a que el formulario en cuestión tan solo tiene un campo de entrada y un botón *submit*, pero para el caso con RIA hay que definir valores etiquetados para definir validaciones de campo a diferencia de MoWebA sin RIA.

Desde el punto de vista práctico, que MoWebA con RIA tarde 8 minutos más que su par sin RIA no es una limitante demasiado condicionante, teniendo en cuenta esos 8 minutos más de modelado permiten a la interfaz de la aplicación *Person Manager* enriquecerse notablemente.

PI2: ¿Para cuál de los enfoques es necesaria una mayor cantidad de generaciones de código para obtener la interfaz de usuario final?

Se pudo apreciar que MoWebA con RIA deparó en una mayor cantidad de generaciones de código para obtener la interfaz RIA final. Esta diferencia representa un aumento del 20% con respecto a la implementación llevada a cabo con MoWebA sin RIA. Analizando las generaciones de cada una de las vistas del *Person Manager* de la Tabla 11 se puede 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 interfaz y 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 MoWebA con RIA, precisamente un 10% más que con el método A, pudiendo deberse también, a que la vista con MoWebA con RIA contiene requerimientos de interfaz RIA a diferencia del método A. También el número de generaciones disminuyó 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 MoWebA con RIA y esto podría deberse a que dada una mayor cantidad de detalles a especificar en los modelos de entrada de la aplicación, existe una mayor posibilidad de cometer fallos.

PI3: Desde el punto de vista de las presentaciones enriquecidas, ¿qué ventajas aportan las características RIA presentes en la aplicación implementada con MoWebA con RIA con respecto a MoWebA sin RIA?

Son diversos los aportes ventajosos de MoWebA con RIA con respecto a MoWebA sin RIA para el *Person Manager*. Las comparativas de las vistas de cada uno de los enfoques puede apreciarse en la Figura 38, Figura 39, Figura 40. A continuación se describen algunas ventajas.

Apariencia de una aplicación single page

Cada una de las páginas que forman parte de la aplicación *Person Manager* implementada con MoWebA con RIA, son equivalentes a las pestañas pertenecientes a un *widget richTab*. 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 MoWebA sin RIA, 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*.

Con la extensión *RichTab* presente en el *Person Manager* con el enfoque RIA, 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 RIA propuesta a MoWebA, como los *RichDatePicker*, *RichToolTip*, *RichAutoSuggest*. La extensión de validación de campos 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 ventajosa con respecto a la optimización espacial de los elementos dentro de las páginas.

Estas mismas ventajas relacionadas al RichTab, también pueden ser aprovechadas al utilizar la extensión *RichAccordion*.

Widgets interactivos en la interfaz de usuario

Todos estos elementos enriquecidos que forman parte del *Person Manager* se han presentado en el capítulo anterior y se pueden apreciar en la Figura 12.

Datepicker

El *datepicker* permite al Person Manager implementado con el enfoque MoWebA con RIA 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 gráfica 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. Para el enfoque MoWebA sin RIA, el campo fecha de nacimiento es solamente un campo de entrada de textual, sin validaciones de formato en la que el usuario debe digitar completamente la fecha de nacimiento. Esto da pie a cometer errores por parte del usuario.

AutoSuggest

El *autoSuggest* ofrece la posibilidad de desplegar un listado de opciones que facilitan al usuario la escritura de texto en un cuadro de texto de entrada. Para el enfoque MoWebA con RIA, 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. En el enfoque MoWebA sin RIA aplicado al *Person Manager*, el campo país de origen es un campo de entrada textual que permite el ingreso de cualquier cadena sin ninguna validación ni sugerencia. En este enfoque el usuario debe escribir el país completamente exponiéndolo a cometer errores.

ToolTip

A menudo es útil complementar con información adicional los campos de entrada de los formularios Web. Con el toolTip en el Person Manager con el enfoque MoWebA con RIA, 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 solicita el ingreso de caracteres alfanuméricos con mayúsculas y minúsculas combinados con caracteres especiales y que contenga por lo menos una longitud de ocho caracteres. En contrapartida, para el enfoque MoWebA sin RIA, no se despliegan mensajes interactivos que podrán complementar a un campo en particular.

PI4: Desde el punto de vista de la lógica de negocios en el lado del cliente, ¿qué ventajas aportan las características RIA presentes en la aplicación implementada con MoWeba con RIA con respecto a MoWebA sin RIA?

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 RIA 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

En el capítulo 4 en la figura 12 se presentan todas las validaciones posibles de campos en el *Person Manager*. 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 *live validation* es posible llevar a cabo validaciones a los diversos campos de los formularios de la aplicación *Person Manager* aplicando el enfoque MoWebA con RIA. Dentro de las validaciones que se han efectuado se muestra primeramente la validación en los campos que son obligatorias y que no pueden quedar vacíos.

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 de 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. En el enfoque aplicado a MoWebA sin RIA, no es posible llevar a cabo validaciones de ninguno de los campos de entrada que forman parte del *Person Manager*. El no contar con ninguna validación tiene implicancias negativas a nivel de seguridad, ya que la aplicación queda expuesta a usos indebidos y errores involuntarios en el ingreso de los datos. También la aplicación queda expuestas al *span*

PI5: Para cada una de las vistas del *Person Manager*, ¿qué cantidad de líneas de código para la interfaz de usuario se pudieron generar de manera automática a partir de los modelos, en cada uno de los enfoques implementados?

Analizando primeramente el tamaño total del *Person Manager* para ambos enfoques, se puede apreciar que el enfoque sin extensiones RIA posee 123 líneas de código menos (equivalente a un 32 %) que el enfoque con extensiones RIA. Esto se debe a que en el enfoque sin RIA no se genera código *Javascript* en la interfaz de usuario ya que su interfaz no posee elementos enriquecidos interactivos.

También puede apreciarse que en el enfoque de MoWebA sin RIA el 47% del código de la aplicación completa fue generado de manera automática a partir de los modelos y el 57% para el caso de MoWebA con RIA. Teniendo en cuenta que el objetivo de este trabajo de fin de carrera está enmarcado en los *front-ends* de las interfaces de usuario web, el porcentaje restante de la aplicación, que fue generado de manera manual (53% y 43%) respectivamente, corresponde a código para refinar la aplicación final y código para el acceso a la capa lógica y de dominio de la aplicación.

Según la definición del *Person Manager* la vista con más requerimientos funcionales enriquecidos corresponde a la vista Agregar Persona en la cual se debe definir *richToolTips*, *richDatepicker*, *richAutoSuggest* y diversas validaciones de campos en el formulario de entrada. Esto conlleva a tener código *Javascript* y *HTML* generado para cada uno de los elementos enriquecidos que han sido definidos en la vista Agregar Persona (135 líneas de código automático y 56 líneas de código manual, lo que indica que el 70% de la interfaz de usuario fue generada de manera automática a partir de los modelos PIM de entrada) para el enfoque de MoWebA con RIA. Sin embargo para la contraparte (MoWebA sin RIA) para la vista Agregar Persona se tiene un 56% menos de código con respecto a MoWebA con RIA, en donde el 47% del código fue generado de manera automática y el 53% agregado de manera manual. Vale la pena acotar que la línea de código manual agregado a cada una de las 3 vistas del Person Manager es el mismo. En vista que el enfoque MoWebA sin RIA no posee elementos interactivos, solamente código *HTML* para cada uno de los elementos definidos es generado, por lo tanto, es natural que existan menos líneas de código.

La vista Mostrar Personas no es muy relevante para el análisis debido a que la extensión propuesta a MoWwbA no contempla el acceso al modelo de dominio, por lo tanto solo el 2% y el 6% del

código es generado de manera automática en cada uno de los enfoques y resto fue implementado de manera manual. Para la vista Remover Persona, en el enfoque sin RIA puede notarse que el 21% del código se genera de manera automática. Esto tiene sentido debido a el formulario que forma parte de esta vista, contiene solamente un campo de entrada sin ningún tipo de validación por lo que es mínimo el código HTML correspondiente al elemento. Sin embargo para el enfoque MoWebA con RIA, esta vista es generada de manera automática en un 58%. Esto tiene su justificativo en que el campo de entrada para esta vista, contiene diversas validaciones, por ende código *Javascript* y *HTML* acompaña al elemento.

En el *Person Manager* se genera código que es común para cada una de las vistas, lo que corresponde a código CSS para representar la parte estructural de la aplicación y código correspondiente al Header HTML con las distintas inclusiones a las librerías jQuery y el archivo CSS. Puede notarse en la tabla que para ambos enfoques bastante de ese código es generado de manera automática (87% para el enfoque MoWebA sin RIA y 58% en su contraparte RIA).

Finalmente se puede concluir de la Tabla 12 que es posible generar más del 50% por ciento de la aplicación final *Person Manager* de manera automática para ambos enfoques.

5.3 RESUMEN DEL CAPITULO

Para ilustrar el uso de la propuesta de extensión a MoWebA con características RIA y para compararla con la propuesta original, un mismo proyecto de ejemplo se implementó con los dos enfoques. Se definieron unas preguntas de investigación para establecer específicamente cuáles son los aspectos a comparar de ambos enfoques. Seguidamente se identificaron las variables a medir para poder contestar las preguntas de investigación. Para las preguntas de investigación que no implicaban una medición (PI3 y PI4), se identificaron cuáles eran las fuentes de información para poder responder a las mismas, concluyendo que con muestras de pantalla de las aplicaciones en cada uno de los enfoques era suficiente.

Luego, durante las implementaciones, se realizaron las mediciones necesarias para las variables de medición asociadas a PI1, PI2 y PI5 y se recolectaron y almacenaron otros datos necesarios para contestar posteriormente las preguntas de investigación.

Finalmente, luego de medir y de recolectar toda la información necesaria para responder las preguntas de investigación, se procedió a analizar estas mediciones y datos recolectados, y se reportaron los resultados obtenidos, respondiendo a cada una de las preguntas de investigación.

CAPITULO 6

CONCLUSIONES Y TRABAJOS FUTUROS

En este trabajo de fin de carrera se llevó a cabo una investigación acerca de las RIA, 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 RIA 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.

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 RIA. 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 RIA 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 RIA a cualquiera de las capas sin afectar a las otras. En este trabajo se extendió con características RIA 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 RIA
- Una investigación del estado del arte de las principales metodologías Web basadas en MDD y MDA que ofrecen cobertura a las RIA.
- Un análisis de los elementos de interfaz enriquecidos (*widgets*) más utilizados en base a un análisis efectuado a portales Web de uso extendido como Facebook, Youtube, Gmail y Amazon.
- Una extensión al metamodelo de contenido de MoWebA agregando los siguiente:
 - o Una reestructuración y clasificación de los elementos de interfaz con los que cuenta la aproximación MoWebA, separando a los distintos componentes de interfaz en elementos de entrada, salida y control.
 - o El agregado del patrón *composite*, para definir una jerarquía entre los distintos elementos de interfaz simples y compuestos

- Widgets interactivos comunes en las aplicaciones RIA, precisamente richAccordion, richTabs, richAutoSuggest, richDatePicker y richToolTip y la validación de diversos campos de entrada.
- Una extensión al metamodelo de estructura Layout, para permitir la definición de las posiciones de los elementos en pixeles o en porcentajes.
- Para la definición de la sintaxis concreta de la presentación, se agregaron los nuevos widgets y la validación de campos al perfil de Contenido, como así también la nueva definición de coordenadas al perfil de Layout 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 destino *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 evaluativa en la cual se presentan los aportes realizados a la capa de presentación de MoWebA.

Finalmente, en la Tabla 13 se presentan las características RIA y a las metodologías web contempladas en el estados del arte junto a la aproximación MoWebA luego de las extensiones RIA.

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	-
	Widgets	si	si	-	si	si	si	si	si	si
	Paradigma de	si	si	si	si	-	si	-	si	si

página única													
		Contenido multimedia	-	sí	-	-	-	-	sí	sí	-	sí	sí
Comunicación cliente servidor	Sincronización de datos	-	-	sí	-	-	-	-	sí	-	sí	-	
	Obtención de actualizaciones parciales de página	sí	sí	sí	sí	sí	-	-	sí	sí	sí	-	
	Push y Pull	-	-	sí	-	-	-	-	sí	-	-	-	

Tabla 13 Tabla ilustrativa presentada en el capítulo 2 junto a la aproximación MoWebA luego de las extensiones RIA.

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 las validaciones locales de los campos, que ofrecen las plataformas jQueryUI³⁷ y jQuery Validation Plugin³⁸.
- Agregar más *widgets* interactivos al metamodelo de contenido como por ejemplo el *dialog*, *menú*, *progressbar*, *selectmenú*, *slider* y *spinner* que también forman parte de jQueryUI.
- 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.
- Validar la propuesta llevando a cabo transformaciones a otras plataformas destino.
- Efectuar la validación de la propuesta de extensión RIA a la capa de presentación de MoWebA, con un caso de estudio formal, detalladamente planeado, 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.

³⁷ jQuery UI 1.11 API Documentation: <http://api.jqueryui.com/> 2015

³⁸ jQuery Validation Plugin: <http://jqueryvalidation.org/> 2015

BIBLIOGRAFÍA

- [1] Ginige A and Murugesan S. Guest editors' introduction: The essence of web engineering—managing the diversity and complexity of web application development. *IEEE MultiMedia*, 8(2):22–25, Apr 2001.
- [2] Wohlin C, Runeson P, Host M, Ohlsson M C, Regnell B, and Wesslén A. *Experimentation in Software Engineering*, volume ISBN 978-3-642-29043-5. Springer, 1 edition, 2012.
- [3] Freeman E, Robson E, Sierra K, and Bates B. *Head first Design Patterns*, volume ISBN 978-0-5960-07126. O'Reilly Media, 2014.
- [4] Valverde F and Pastor O. Applying interaction patterns. In *Towards a Model-Driven Approach for Rich Internet Applications Development. Proc. 7th Int. Workshop. on Web-Oriented Software technologies, IWWOST 2008*, 2008.
- [5] Martínez-Ruiz F J, Arteaga J M, Vanderdonckt J, Gonzalez-Calleros. J M, and Mendoza R. A first draft of a model-driven method for designing graphical user interfaces of rich internet applications. In *Proceedings of the Fourth Latin American Web Congress, LA-WEB '06*, pages 32–38, Washington, DC, USA, 2006. IEEE Computer Society.
- [6] Martínez-Ruiz F J. *A Development Method for User Interfaces of Rich Internet Applications*. PhD thesis, Polytechnic University of Valencia, Spain, August 2010.
- [7] Toffetti G, Comai S, Preciado J C, and Linaje M. State-of-the art and trends in the systematic development of rich internet applications. *J. Web Eng.*, 10(1):70–86, March 2011.
- [8] Allaire J. Requirements for rich internet applications. <http://download.macromedia.com/pub/flash/whitepapers/richclient.pdf>, March 2002.
- [9] Gómez J, Cachero C, and Pastor O. Extending an object-oriented conceptual modeling approach to web application design. In *In Proceedings of the 12th International Conference on Advanced Information Systems Engineering*, pages 79–93. Springer-Verlag, 2000.
- [10] Wright J and Dietrich J. Survey of existing languages to model interactive web applications. In *Proceedings of the fifth Asia-Pacific conference on Conceptual Modelling - Volume 79*, APCCM '08, pages 113–123, Darlinghurst, Australia, Australia, 2008. Australian Computer Society, Inc.
- [11] Preciado J C, Linaje M, Sanchez F, and Comai S. Necessity of methodologies to model rich internet applications. In *Proceedings of the Seventh IEEE International Symposium on Web Site Evolution, WSE '05*, pages 7–13, Washington, DC, USA, 2005. IEEE Computer Society.
- [12] Preciado J C, Linaje M, Morales-Chaparro R, Sanchez-Figueroa F, Zhang G, Kroiß C, and Koch N. Designing rich internet applications combining uwe and rux-method. In *Proceedings of the*

2008 Eighth International Conference on Web Engineering, ICWE '08, pages 148–154, Washington, DC, USA, 2008. IEEE Computer Society.

- [13] Garrett J. J. Ajax: A new approach to web applications. <http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications/>, February 2005.
- [14] H Kiewe. Rich internet applications: What's the business case? <http://es.slideshare.net/hkiewe/rich-internet-applications-whats-the-business-case-howard-kiewe-at-ajaxworld-west>, 2011.
- [15] Machado L, Filho O, and Ribeiro J. Uwe-r: an extension to a web engineering methodology for rich internet applications. *WSEAS Trans. Info. Sci. and App.*, 6(4):601–610, April 2009.
- [16] Brambilla M, Cabot J, and Wimmer M. *Model-Driven Software Engineering in Practice*, volume ISBN 9781608458820. Morgan & Claypool, 2012.
- [17] Busch M and Koch N. Rich internet applications state-of-the-art. Technical report 0902, Programming and Software Engineering Unit (PST), Institute for Informatics, Ludwig-Maximilians-Universität München, Germany, December 2009.
- [18] González M, Casariego J, Bareiro J, Cernuzzi L, and Pastor O. Una propuesta mda para las perspectivas navegacional y de usuarios. In *XXXVI Conferencia Latinoamericana de Informática (CLEI)* - ISBN 978-99967-612-0-1, page 58, Asunción, Paraguay, 2010.
- [19] González M, Cernuzzi L, and Pastor O. Una aproximación para aplicaciones web: Moweba. In *XIV Congreso Iberoamericano en Software Engineering – CibSE*, Río de Janeiro, Brasil, 2011.
- [20] Linaje-Trigueros M, Preciado J C, and Sanchez-Figueroa F. Engineering rich internet application user interfaces over legacy web models. *IEEE Intern. Comput.*, 11(6):53–59, November 2007.
- [21] Urbieta M, Rossi G, Ginzburg J, and D. Schwabe. Designing the interface of rich internet applications. In *Proceedings of the 2007 Latin American Web Conference*, LA-WEB '07, pages 144–153, Washington, DC, USA, 2007. IEEE Computer Society.
- [22] Koch N. Hypermedia systems development based on the unified process. Tech. rept, Ludwig-Maximilians- Universität München, Germany, 2000.
- [23] Koch N. *Software Engineering for Adaptive Hypermedia Systems, Reference Model, Modeling Techniques and Development Process*. PhD thesis, Verlag UNI-DRUCK, ISBN 3-87821-318-2, 2001.
- [24] Koch N, Pigerl M, Zhang G, and Morozova T. Patterns for the model-based development of rias. In *Proceedings of the 9th International Conference on Web Engineering*, ICWE '9, pages 283–291, Berlin, Heidelberg, 2009. Springer-Verlag.

- [25] Dolog P and Stage J. Designing interaction spaces for rich internet applications with uml. In *Proceedings of the 7th International Conference on Web Engineering*, ICWE'07, pages 358–363, Berlin, Heidelberg, 2007. Springer-Verlag.
- [26] Fraternali P, Comai S, Bozzon A, and Carughi G T. Engineering rich internet applications with a model-driven approach. *ACM Trans. Web*, 4(2):7:1–7:47, April 2010.
- [27] Runeson P, Höst M, Rainer A, and Regnell B. *CASE STUDY RESEARCH IN SOFTWARE ENGINEERING. Guidelines and Examples*, volume ISBN 978-1118104354. Jhon Wiley & Sons,Inc, Hoboken, New Jersey, 2012.
- [28] Wieringa R. *Design Science Methodology for Information Systems and Software Engineering*, volume ISBN-978-3-662-43838-1. Springer, Springer Heidelberg New York Dordrecht London, 2014.
- [29] R Rogowski. The business case for rich internet applications. http://www.adobe.com/engagement/pdfs/bus_case_ria.pdf, March 2007.
- [30] Meliá S, Gómez J, Pérez S, and Dáz O. A model-driven development for gwt-based rich internet applications with ooh4ria. In *Proceedings of the 2008 Eighth International Conference on Web Engineering*, ICWE '08, pages 13–23, Washington, DC, USA, 2008. IEEE Computer Society.
- [31] Vahid Gharavi S V. Model-driven development of ajax web applications. Master's thesis, Faculty EEMCS, Delft University of Technology, September 2008.

ANEXO 1

UN SISTEMA DE GESTIÓN PERSONAS - PERSON MANAGER

DEFINICIÓN

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 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, los datos ingresados se insertan en una base de datos. La vista Agregar Persona debe permitir ingresar los campos; nombre, apellido, fecha de nacimiento, país de origen, email, usuario, clave, confirmación de clave; como así también se debe poder seleccionar el sexo de la persona y verificar que los datos insertados han sido correctos.

Listar Personas (List): Consiste en una vista en la cual es posible desplegar 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 Persona (Remove): se trata de una vista para borrar a una persona de la base de datos. En un formulario, el id de la persona a borrar es ingresado. 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 Personas.

Adicionalmente se adicionan los siguientes requerimientos RIA:

Para los campos de la vista Agregar Persona:

- Para el campo fecha de nacimiento de la persona, se desea que el ingreso de la fecha sea ágil, intuitivo e interactivo y que no sea necesario escribir la fecha manualmente.
- En ciertos campos como nombre, apellido y clave, se requiere información adicional interactiva que guíe al usuario a ingresar nombre y apellido completo y una clave segura de al menos 8 caracteres.
- Para el campo país de origen se busca que el usuario escriba la menor cantidad caracteres posibles, por lo tanto a medida que se introducen caracteres correspondientes a un país, el sistema debe desplegar sugerencias en base al patrón actual ingresado por el usuario, permitiéndole navegar en tales sugerencias, hasta elegir la opción deseada.

- Las siguientes validaciones de campos deben llevarse a cabo:
 - **Validaciones de campos obligatorios:** Para que el formulario pueda ser validado, los campos nombre, apellido, clave, y confirmar clave, deben ser completados de manera obligatoria, también, para el género, debe seleccionarse una opción de las existentes (masculino o femenino). Finalmente, una confirmación de acuerdo sobre los datos introducidos debe marcarse.
 - El sistema debe informar al usuario en caso de que se ingrese un nombre de usuario incorrecto o una clave insegura, partiendo de las siguientes recomendaciones:
 - Para el campo usuario, la longitud debe ser de al menos tres caracteres.
 - Para el campo clave, la longitud mínima debe ser de 8 caracteres.
 - El sistema debe informar al usuario en caso de insertar un email con formato erróneo.
 - El sistema debe alertar al usuario en caso que los campos clave y confirmación de clave no coincidan.

Para el campo de la vista Eliminar Persona:

- Se requiere que el sistema valide que el campo id de la persona contenga solamente valores enteros no negativos y que no supere los 10 dígitos. El campo es de ingreso obligatorio y no puede quedar vacío.

De forma general, se espera que el sistema tenga la apariencia de una aplicación de escritorio. Por lo tanto la interfaz de usuario debe ofrecer un aspecto single page, en donde la navegación por las distintas vistas de la aplicación no debe implicar un refresh 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 obligatorio, 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*.

ANEXO 2

PLANTILLA DE TRANSFORMACION PARA LOS PIM DE CONTENIDO

```
[comment encoding = UTF-8 /]

[module generate('http://www.eclipse.org/uml2/4.0.0/UML')]
[import org::eclipse::acceleo::module::presentation::services::uml2services /]

[template public generateElement(c : Class)
[comment @main/]
[if c.hasStereotype('presentationPage') ]
    [file (c.name.concat('.html'), false, 'UTF-8')]
<!doctype html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>[c.name.toString()]/</title>
<link rel="stylesheet" href="jquery-ui-1.11.3.custom/jquery-ui.min.css">
<script src="jquery-ui-1.11.3.custom/external/jquery/jquery.js"></script>
<script src="jquery-ui-1.11.3.custom/jquery-ui.min.js"></script>

<link rel="stylesheet" href="jquery-validation-1.13.1/demo/css/screen.css">
<script src="jquery-validation-1.13.1/dist/jquery.validate.js"></script>
</head>
<script>
    [comment codigo js para el widget acordeon /]
    [for (c2: Class | c.followingSiblings(Class))]
        [for (s: Stereotype | c2.getAppliedStereotypes()->asSequence())]
            [if s.name.toString() = 'richAccordion']
$(function() {
    $('#[c2.name.substituteAll(' , '')/]').accordion();
});
    [/if]
    [/for]
    [/for]
    [comment codigo js para el widget tab /]
    [for (c2: Class | c.followingSiblings(Class))]
        [for (s: Stereotype | c2.getAppliedStereotypes()->asSequence())]
            [if s.name.toString() = 'richTabs']
$(function() {
    $('#[c2.name.substituteAll(' , '')/]').tabs();
});
    [/if]
    [/for]
    [/for]
    [comment codigo js para el widget fieldLiveValidation
    (para validacion de mÃ¡s de 1 campo. Primeramente la definiciÃ³n de la
    funciÃ³n principal y sus reglas.
    Primeramente las reglas/]
    [for (c2: Class | c.followingSiblings(Class))]
        [for (s1: Stereotype | c2.getAppliedStereotypes()->asSequence())]
            [if s1.name.toString() = 'form' and c2.getValue(s1, 'validate') = true]
$().ready(function() {
    // validate signup form on keyup and submit
    $('#[c2.name.substituteAll(' , '')/]').validate({
rules: {
    [for (att: Property | c2.getAllAttributes()->asSequence())]
        [for (s: Stereotype | att.getAppliedStereotypes()->asSequence())]

            [if s.name.toString() = 'richFieldLiveValidation']
                [comment Primeramente validacion tipo password
]
                [if att.getValue(s,
'validation').toString().strstr('password') = true ]
                    [att.name/]: {
                        required: true,
                        [if att.getValue(s,'required') = true]
                            [/if]
                            [if att.getValue(s,'minLength').toString().toInteger() <> 0]
                                minlength: [att.getValue(s,'minLength').toString().toInteger() /],
                                [/if]
                                [if att.getValue(s,'maxLength').toString().toInteger() <> 0]
                                    maxlength: [att.getValue(s,'maxLength').toString().toInteger() /],
                                    [/if]
}
},
```

```

        [/if]
        [if att.getValue(s,'confirmPass') = true]
        [if att.getValue(s,'required') = true]
    [/if]
    [if]
    [if att.getValue(s,'minLength').toString().toInteger() <> 0]
    minlength: [att.getValue(s,'minLength')],
    [/if]
    [if]
    [if att.getValue(s,'maxLength').toString().toInteger() <> 0]
    maxlength: [att.getValue(s,'maxLength').toString().toInteger() /],
    [/if]
    equalTo: "#[att.name]"
},
[elseif att.getValue(s,
'validation').toString().strstr('email') = true]
[if att.getValue(s,'required') =
true]
required: true,
[/if]
[if
[if att.getValue(s,'minLength').toString().toInteger() <> 0]
minlength: [att.getValue(s,'minLength')],
[/if]
[if
[if att.getValue(s,'maxLength').toString().toInteger() <> 0]
maxlength: [att.getValue(s,'maxLength').toString().toInteger() /],
[/if]
email: true
},
[elseif att.getValue(s,
'validation').toString().strstr('simpleTextImput') = true]
[if att.getValue(s,'required') =
true]
required: true,
[/if]
[digits: true,
[if
[if att.getValue(s,'digits') = true]
[/if]
[if
[if att.getValue(s,'minLength').toString().toInteger() <> 0]
minlength: [att.getValue(s,'minLength')],
[/if]
[if
[if att.getValue(s,'maxLength').toString().toInteger() <> 0]
maxlength: [att.getValue(s,'maxLength').toString().toInteger() /],
[/if]
],
[elseif att.getValue(s,
'validation').toString().strstr('agreeCheck') = true]
agree: "required"
[/if]
[/for]
[/for]
[/if]
[/for]
[/for] [comment Fin de la generaciÃ³n de las reglas para el fieldLiveValidation (para validacion de mÃ¡s de un
campo /]
[comment codigo js para el widget fieldLiveValidation (para validacion
de mÃ¡s de 1 campo. DefiniciÃ³n de las reglas y cierre de la funciÃ³n
principal./]
[for (c2: Class | c.followingSiblings(Class))
    [for (s1: Stereotype | c2.getAppliedStereotypes()->asSequence())
        [if s1.name.toString() = 'form' and c2.getValue(s1, 'validate') = true]
messages: {
    [for (att: Property | c2.getAllAttributes()->asSequence())
        [for (s: Stereotype | att.getAppliedStereotypes()->asSequence())
            [if s.name.toString() = 'richFieldLiveValidation'
                [if att.getValue(s,
'validation').toString().strstr('password') = true ]
                [if att.getValue(s,'required') =
true]
                required: "Por favor ingrese un [att.name.toUpperCaseFirst()]/i",
                [/if]
                [if att.getValue(s,'minLength').toString().toInteger() <> 0]
                minlength: "El [att.name] debe tener al menos [att.getValue(s,'minLength')/] caracteres de longitud",
                [/if]
                [if att.getValue(s,'maxLength').toString().toInteger() <> 0]
                maxlength: "El [att.name] debe tener como maximo [att.getValue(s,'maxLength')/] caracteres de longitud",
                [/if]
            ]
        ]
    ]
}

```

```

    },
    [/if]
    [if att.getValue(s,'confirmPass') = true]
        [if att.getValue(s,'required') = true]
            required: "Por favor ingrese un [att.name.toUpperCase()]",

            [/if]
            [if att.getValue(s,'minLength').toString().toInteger() <>
0]
                minlength: "El [att.name/] debe tener al menos [att.getValue(s,'minLength')/] caracteres de longitud",
                [/if]
                [if att.getValue(s,'maxLength').toString().toInteger() > 0]
                    maxlength: "El [att.name/] debe tener como maximo [att.getValue(s,'maxLength')/] caracteres de longitud",
                    [/if]
            equalTo: "Por favor ingrese el mismo password"
        },
        [elseif att.getValue(s,
'validation').toString().strstr('email') = true ]
        [att.name/]: {
            [if att.getValue(s,'required') =
true]
                required: "Por favor ingrese un [att.name.toUpperCase()]",

                [/if]
                [if att.getValue(s,'minLength').toString().toInteger() <> 0]
                    minlength: "El [att.name/] debe tener al menos [att.getValue(s,'minLength')/] caracteres de longitud",
                    [/if]
                    [if att.getValue(s,'maxLength').toString().toInteger() <> 0]
                        maxlength: "El [att.name/] debe tener como maximo [att.getValue(s,'maxLength')/] caracteres de longitud",
                        [/if]
                email: "Por favor ingrese una direcciÃ³n de email valida"
            },
            [elseif att.getValue(s,
'validation').toString().strstr('simpleTextInput') = true ]
            [att.name/]: {
                [if att.getValue(s,'required') = true]
                    required: "Por favor ingrese un [att.name.toUpperCase()]",

                    [/if]
                    [if att.getValue(s,'minLength').toString().toInteger() <> 0]
                        minlength: "El [att.name/] debe tener al menos [att.getValue(s,'minLength')/] caracteres de longitud",
                        [/if]
                        [if att.getValue(s,'maxLength').toString().toInteger() <> 0]
                            maxlength: "El [att.name/] debe tener como maximo [att.getValue(s,'maxLength')/] caracteres de longitud",
                            [/if]
                        digits: "El [att.name/] debe contener digitos",
                        [/if]
                },
                [elseif att.getValue(s, 'validation').toString().strstr('agreeCheck') =
true ]
                agree: "Favor seleccione este campo"
                [/if]
                [/for]
            },
            [/for]
        },
        [/if]
        [/for] [comment Fin de la generaciÃ³n de los mensajes y la funciÃ³n principal para el fieldLiveValidation (para
validacion de mÃ¡s de un campo /]

        [comment codigo js para los widget autoSuggest, datepicker tooltip/]
        [for (c2: Class | c.followingSiblings(Class))]
            [for (att: Property | c2.getAllAttributes()->asSequence())]
                [for (s: Stereotype | att.getAppliedStereotypes()->asSequence())]
                    [if s.name.toString() = 'richAutoSuggest']

        $(function() {
            [if att.getValue(s,
'localDictionary').toString().strstr('definedInTheModel') = true]
                var availableTags[att.name.toUpperCase()/] =
                    ['[ / ]'[att.getValue(s, 'source')]/[' / ];
                $( "#[att.name.replaceAll(' ', '')/]" ).autocomplete({
                    source: availableTags[att.name.toUpperCase()]
                });
            },
            [elseif att.getValue(s, 'localDictionary').toString().strstr('definedInAFile') =
true]
            $.ajax({
                url: "[att.getValue(s, 'source')/]",
                dataType: "xml",
                success: function( xmlResponse ) {
                    var data = $( "tag", xmlResponse ).map(function() {
                        return {
                            value: $( "name", this ).text(),
                            id: $( "tagId", this ).text()
                        };
                }
            }
        )
    
```

```

});.get();
$( "#[att.name.substituteAll(' ', '')/]" ).autocomplete({
    source: data,
});
});
[/if]
});

$(function() {
$( "#[att.name.substituteAll(' ', '')/]" ).datepicker({
    changeYear: true,
    [/if]
    [if att.getValue(s,'changeMonth') = true]
    changeMonth: true,
    [/if]
    [if att.getValue(s,'yearRange').toString().size() <> 0]
    yearRange: "[att.getValue(s, 'yearRange').toString()/"],
    [/if]
    [if att.getValue(s, 'dateFormat').toString().strstr('Default - mm/dd/yy') = true
]
    dateFormat: "mm/dd/yy"
    [elseif att.getValue(s, 'dateFormat').toString().strstr('ISO 8601 - yy-mm-dd') = true ]
    dateFormat: "yy-mm-dd"
    [elseif att.getValue(s, 'dateFormat').toString().strstr('Short - d M, y') = true ]
    dateFormat: "d M, y"
    [elseif att.getValue(s, 'dateFormat').toString().strstr('Medium - d
MM, y') = true ]
    dateFormat: "d MM, y"
    [elseif att.getValue(s, 'dateFormat').toString().strstr('Full - DD, d
MM, yy') = true ]
    dateFormat: "DD, d MM, yy"
    [/if]
});
});
[elseif s.name.toString() = 'richToolTip']
$( document ).tooltip();
});
[/if]
[/for]
[/for] [comment fin de la generaciÃ³n .js para el richAutoSuggest, richDatePicker y el richToolTip /]
</script> [comment fin de la generaciÃ³n del codigo .js /]

[comment generaciÃ³n del body para el html /]
<body>
[comment generaciÃ³n de contenido para el richAccordion /]
[for (c2: Class | c.followingSiblings(Class))]
    [if c2.hasStereotype('richAccordion') ]
<div id="[c2.name.substituteAll(' ', '')/]">
    [for (s1: Stereotype | c2.getAppliedStereotypes()->asSequence())]
    [for (c3: Class | c2.followingSiblings(Class))]
        [if c3.getValue(s1, 'content').toString().strstr('withinARichAccordion')]
<h3>[c3.name.toUpperCaseFirst()]</h3>
<div>
    [/if]
    [if c3.hasStereotype('compositeUIElement') and c3.hasStereotype('form') and c3.getValue(s1,
'content').toString().strstr('withinARichAccordion')]
<form id="[c3.name.substituteAll(' ', '')/]" method="POST" action="" ">
    [for (att: Property | c3.getAllAttributes()->asSequence())]
        [for (s: Stereotype | att.getAppliedStereotypes()->asSequence())]
            [if s.name.toString() = 'richAutoSuggest']

<p>
    <label for="[att.name.substituteAll(' ', '')/]">[att.name.toUpperCaseFirst()]</label>
    <input id="[att.name.substituteAll(' ', '')/]" type="text" name="[att.name.substituteAll(' ', '')/]">
</p>
[/if] [comment fin del richAutoSuggest /]
[if s.name.toString() = 'richDatePicker']

<p>[att.name.toUpperCaseFirst()] <input type="text" id="[att.name.substituteAll(' ', '')/]" name="[att.name.substituteAll(' ', '')/]" [if att.getValue(s, 'title').toString().size() <> 0] title="[att.getValue(s, 'title').toString()]/"[/if]></p>

    [/if] [comment fin del richDatePicker /]
    [if s.name.toString() = 'richFieldLiveValidation']
        [comment validacion del tipo password /]
        [if att.getValue(s, 'validation').toString().strstr('password') = true ]

<p>
    <label for="[att.name.substituteAll(' ', '')/]">[att.name.toUpperCaseFirst()]</label>
    <input id="[att.name.substituteAll(' ', '')/]" name="[att.name.substituteAll(' ', '')/]" type="password" [if
att.getValue(s, 'title').toString().size() <> 0] title="[att.getValue(s, 'title').toString()]/"[/if]><br>
</p>
    [comment si el tagged value confirmPass esta definido /]
    [if att.getValue(s, 'confirmPass') = true]

<p>
    <label for="confirm_[att.name/]">Confirm [att.name.toUpperCaseFirst()]</label>
    <input id="confirm_[att.name.substituteAll(' ', '')/]" name="[att.name.substituteAll(' ', '')/]" type="password"

```

```

        </p>
        [</if>
         [comment validacion del tipo email /]
         [elseif att.getValue(s, 'validation').toString().strstr('email') = true ]
      <p>
        <label for="[att.name.substringAll(' ', '')/]">[att.name.toUpperCase()]/</label>
        <input id="[att.name.substringAll(' ', '')/]" name="[att.name.substringAll(' ', '')/]" type="email" [if
att.getValue(s, 'title').toString().size() <> 0] title=[att.getValue(s, 'title').toString()]/</if><br>
      </p>
        [comment validacion de un textInput simple /]
        [elseif att.getValue(s, 'validation').toString().strstr('simpleTextInput') = true ]
      <p>
        <label for="[att.name.substringAll(' ', '')/]">[att.name.toUpperCase()]/</label>
        <input id="[att.name.substringAll(' ', '')/]" name="[att.name.substringAll(' ', '')/]" type="text" [if
att.getValue(s, 'title').toString().size() <> 0] title=[att.getValue(s, 'title').toString()]/</if><br>
      </p>
        [comment validacion del tipo agree checkbox /]
        [elseif att.getValue(s, 'validation').toString().strstr('agreeCheck') = true ]
      <p>
        <label for="[att.name.substringAll(' ', '')/]">[att.name.toUpperCase()]/</label>
        <input type="checkbox" id="[att.name.substringAll(' ', '')/]" name="[att.name.substringAll(' ', '')/]>
      </p>
        [comment validacion del tipo radio choice/]
        [elseif att.getValue(s, 'validation').toString().strstr('radioChoice') = true ]
      <fieldset>
        <legend>[att.name.substringAll(' ', '').toUpperCase()]/</legend>
        [for (a : String | att.getValue(s, 'choiceOptions').toString().tokenize(',') ) ]
        <label for="["<if a.strstr('Required')]>[a.replace(' Required','')/][else]<a>[a.toString().substringAll(' ', ''/)]</if>">
          <input type="radio" id="["<if a.strstr('Required')]>[a.replace(' Required','')/][else]<a>[a/]</if>
name="["<att.name.substringAll(' ', '')/]" value="["<if a.strstr('Required')]>[a.replace(' Required','')/][else]<a>[a/]</if> [<if
a.strstr('Required')]>required</if>["<if a.strstr('Required')]>[a.toUpperCase().replace(' Required','')/][else]<a>[a.toUpperCase()]/</if>
</label>
          [</for>
        </fieldset>
        [</if>
        [</if> [comment fin del 'richFieldLiveValidation' /]
        [if s.name.toString() = 'textInput']
        <label for="["<att.name.substringAll(' ', '')/]">[att.name.toUpperCase()]/</label><input id="["<att.name.substringAll(' ', '')/]" name="["<att.name.substringAll(' ', '')/]" [if att.getValue(s, 'title').toString().size() <> 0] title=[att.getValue(s, 'title').toString()]/>["<if>[if att.getValue(s, 'password') = false] type="text" [else] type="password" [<if> [if att.getValue(s,
'mandatory') = true] required [</if>]<br>
          [</if>
          [if s.name.toString() = 'htmlText']
          [<att.getValue(s, 'html')/]>
          [</if> [comment fin del htmlText' /]
          [if s.name.toString() = 'multimedia']
            [if att.getValue(s, 'type').toString().strstr('video') = true ]
            [if att.getValue(s, 'url').toString().size() <> 0 ]
            [</if>
            [</if>
              <object width="420" height="315"
                data="["<att.getValue(s, 'url')/]">
            </object>
            [</if>
            [if att.getValue(s, 'path').toString().size() <> 0 ]
            <video width="320" height="240" controls>
              <source src="["<att.getValue(s, 'path')/]"> [if att.getValue(s, 'path').toString().strstr('.mp4') = true]
            type="video/mp4">
            [else] type="video/ogg"> [</if>
            Your browser does not support the video tag.
          </video>
          [</if>
          [</if>
            [if att.getValue(s, 'type').toString().substr('image') = true ]
            [if att.getValue(s, 'url').toString().size() <> 0 ]
            
            [</if>
            [if att.getValue(s, 'path').toString().size() <> 0 ]
            
            [</if>
            [</if>
            [if att.getValue(s, 'type').toString().substr('sound') = true ]
            [if att.getValue(s, 'path').toString().size() <> 0 ]
            <audio controls>
              <source src="["<att.getValue(s, 'path')/]"> [if att.getValue(s, 'path').toString().substr('.mp3') = true]
            type="audio/mpeg">
            [else] type="audio/ogg"> [</if>
            Your browser does not support the audio element.
          </audio>
          [</if>
          [</if>
            [if s.name.toString() = 'externalLink']
            [</if>
              <a href="["<att.getValue(s, 'url')/]"> [if att.getValue(s, 'title').toString().size() <> 0 ] title=[att.getValue(s, 'title')/]">["<att.name.toUpperCase()/"></a>
            [</if>

```

```

        [/if] [comment fin del externalLink /]
        [if s.name.toString() = 'anchor']
    <a href=[att.getValue(s, 'virtualStateName').toString().toUpperCase().concat('.php')/]> [if att.getValue(s,
'title').toString().size() > 0 ] title=[att.getValue(s, 'title')/][/if][att.name.toUpperCase()]/</a>
        [if] [comment fin del anchor /]
        [if s.name.toString() = 'text']
    <p[if att.getValue(s, 'title').toString().size() > 0 ] title=[att.getValue(s, 'title')/][/if]>[att.getValue(s,
'localText')/]</p>
        [if]
            [if s.name.toString() = 'button']
        <p>
            <input type="submit" value=[att.name.toUpperCase()]/>
        </p>
        [if]
            [if s.name.toString() = 'list']
                [if att.getValue(s, 'listType').toString().strchr('choice') = true ]
                    [if att.getValue(s, 'filter').toString().size() > 0 ]
            <fieldset>
                <legend>[att.name/]</legend>

                <input type="radio" id=[a/] value=[a/] name=[att.name/]> [a.toUpperCase()]/
                [for (a : String | att.getValue(s, 'filter').toString().tokenize(',') ) ]
                </input>
            </fieldset>
            [if] <br>
            [if]
                [if att.getValue(s, 'listType').toString().strchr('dropBox') = true ]
                    [if att.getValue(s, 'filter').toString().size() > 0 ]
                <p>[att.getValue(s, 'title')/]<input list=[att.getValue(s, 'title').toString().substituteAll(' ', '')/]>
name=[att.getValue(s, 'title').toString().substituteAll(' ', '')/]>
                <datalist id=[att.getValue(s, 'title').toString().substituteAll(' ', '')/]>

                    [for (a : String | att.getValue(s, 'filter').toString().tokenize(',') ) ]
                    <option value=[a/]>
                </for>
            </p>
            [if] <br>
            [if]
                [if]
                    [if]
                        [if]
                            [for (a : String | att.getValue(s, 'filter').toString().tokenize(',') ) ]
                            <option value=[a/]>
                        </for>
                    </if>
                </if>
            </if>
        </form>
        [elseif c3.hasStereotype('compositeUIElement') and c3.hasStereotype('form')=false and c3.getValue(s1,
'content').toString().strchr('withinARichAccordion'))]
        [for (att: Property | c3.getAllAttributes()->asSequence())]
            [for (s: Stereotype | att.getAppliedStereotypes()->asSequence())]
                [if s.name.toString() = 'htmlText']
                    [if] [comment fin del htmlText' /]
                    [if s.name.toString() = 'multimedia']
                        [if att.getValue(s, 'type').toString().strchr('video') = true ]
                            [if att.getValue(s, 'url').toString().size() > 0 ]

<object width="420" height="315"
    data=[att.getValue(s, 'url')/]>
</object>
        [if]
            [if att.getValue(s, 'path').toString().size() > 0]
<video width="320" height="240" controls>
    <source src=[att.getValue(s, 'path')/]> [if att.getValue(s, 'path').toString().strchr('.mp4') = true ] type="video/mp4">
        [else] type="video/ogg"> [if]
            Your browser does not support the video tag.
</video>
        [if]
        [if]
            [if att.getValue(s, 'type').toString().strchr('image') = true ]
                [if att.getValue(s, 'url').toString().size() > 0 ]

<img src=[att.getValue(s, 'url')/]" alt="W3Schools.com" style="width:104px;height:142px">
        [if]
            [if att.getValue(s, 'path').toString().size() > 0]
<img src=[att.getValue(s, 'path')/]" alt="HTML5 Icon" style="width:400px;height:400px">
        [if]
            [if att.getValue(s, 'type').toString().strchr('sound') = true ]
                [if att.getValue(s, 'path').toString().size() > 0]
<audio controls>
    <source src=[att.getValue(s, 'path')/]> [if att.getValue(s, 'path').toString().strchr('.mp3') = true ] type="audio/mpeg">
        [else] type="audio/ogg"> [if]
            Your browser does not support the audio element.
</audio>
        [if]
        [if]
            [if s.name.toString() = 'externalLink']

```

```

<a href="[att.getValue(s, 'url')]" [if att.getValue(s, 'title').toString().size() <> 0 ] title="[att.getValue(s, 'title')]">
[/if]>[att.name.toUpperCase()]/</a>
[if] [comment fin del externalLink /]
[if s.name.toString() = 'anchor']
<a href="[att.getValue(s, 'virtualStateName').toString().toUpperCase().concat('.php')]" [if att.getValue(s,
'title').toString().size() <> 0 ] title="[att.getValue(s, 'title')]" [if]>[att.name.toUpperCase()]/</a>
[/if] [comment fin del anchor /]
[if s.name.toString() = 'text']
<p[if att.getValue(s, 'title').toString().size() <> 0 ] title="[att.getValue(s, 'title')]" [if]>[att.getValue(s,
'localText')]/</p>
[/if]
[if s.name.toString() = 'list']
[if att.getValue(s, 'listType').toString().strStr('choice') = true ]
[if att.getValue(s, 'filter').toString().size() <> 0 ]
<fieldset>
<legend>[att.name]/</legend>

[for (a : String | att.getValue(s, 'filter').toString().tokenize(',') ) ]
<input type="radio" id="[a]" value="[a]" name="[att.name]"/> [a.toUpperCase()]/
[/for]
</fieldset>
[/if] <br>
[/if]
[if att.getValue(s, 'listType').toString().strStr('dropBox') = true ]
[if att.getValue(s, 'filter').toString().size() <> 0 ]
<p>[att.getValue(s, 'title')]/<input list="[att.getValue(s, 'title').toString().substituteAll(' ', '')]"/>
name="[att.getValue(s, 'title').toString().substituteAll(' ', '')]"/>
<datalist id="[att.getValue(s, 'title').toString().substituteAll(' ', '')]"/>

[for (a : String | att.getValue(s, 'filter').toString().tokenize(',') ) ]
<option value="[a]"/>
[/for]
</datalist>
<p>
[/if] <br>
[/if]
[/if]
[/for]
[/if]
[if c3.getValue(s1, 'content').toString().strStr('withinARichAccordion')]

</div>
[/if]
[/for]
[/for]
</div>
[/if]
[/for] [comment fin de la generación de contenido para el richAccordeon /]
[comment generación de contenido para el richTabs - Las pestañas /]
<for (c2: Class | c.followingSiblings(Class))>
[if c2.hasStereotype('richTabs')]
<div id="[c2.name.substituteAll(' ', '')]"/>
<ul>
<for (s1: Stereotype | c2.getAppliedStereotypes()->asSequence())>
<for (c3: Class | c2.followingSiblings(Class)->asSequence())>
[if c3.getValue(s1, 'content').toString().strStr('withinARichTab')]
<li><a href="#[c3.name.substituteAll(' ', '')]"/>[c3.name.toUpperCase()]/</a></li>
[/if]
[/for]
[/for]
</ul>
[/if]
[/for] [comment fin generación de contenido para el richTabs - Las pestañas /]
[comment generación de contenido para el richTabs - El contenido de las pestañas /]
<for (c2: Class | c.followingSiblings(Class))>
[if c2.hasStereotype('richTabs')]
[/for (s1: Stereotype | c2.getAppliedStereotypes()->asSequence())>
<for (c3: Class | c2.followingSiblings(Class))>
[if c3.getValue(s1, 'content').toString().strStr('withinARichTab')]
<div id ="[c3.name.substituteAll(' ', '')] ">
[/if]
[if c3.hasStereotype('compositeUIElement') and c3.hasStereotype('form') and c3.getValue(s1,
'content').toString().strStr('withinARichTab')]
<form id="[c3.name.substituteAll(' ', '')]"/> method="POST" action="" " >
<for (att: Property | c3.getAllAttributes()->asSequence())>
[for (s: Stereotype | att.getAppliedStereotypes()->asSequence())>
[if s.name.toString() = 'richAutoSuggest']

<p>
<label for="[att.name.substituteAll(' ', '')]"/>[att.name.toUpperCase()]/</label>
<input id="[att.name.substituteAll(' ', '')]"/> type="text" name="[att.name.substituteAll(' ', '')]"/>
</p>
[/if] [comment fin del richAutoSuggest /]
[if s.name.toString() = 'richDatePicker']

<p>[att.name.toUpperCase()]/<input type="text" id="[att.name.substituteAll(' ', '')]"/> name="[att.name.substituteAll(' ', '')]"/> [if att.getValue(s, 'title').toString().size() <> 0 ] title="[att.getValue(s, 'title').toString()]/" [if]></p>
[/if] [comment fin del richDatePicker /]
[if s.name.toString() = 'richFieldLiveValidation']
[comment validacion del tipo password /]
```

```

[if att.getValue(s, 'validation').toString().strstr('password') = true ]

<p>
    <label for="[att.name.replaceAll(' ', '')]">[att.name.toUpperCase()]/</label>
    <input id="[att.name.replaceAll(' ', '')]" name="[att.name.replaceAll(' ', '')]" type="password" [if
att.getValue(s, 'title').toString().size() > 0] title="[att.getValue(s, 'title').toString()]/[/if]><br>
</p>
    [comment si el tagged value confirmPass esta definido /]
    [if att.getValue(s, 'confirmPass') = true]

<p>
    <label for="confirm_[att.name]">Confirm [att.name.toUpperCase()]/</label>
    <input id="confirm_[att.name.replaceAll(' ', '')]" name="[att.name.replaceAll(' ', '')]" type="password" [if
type="password"
</p>
    [if]
    [comment validacion del tipo email /]
    [elseif att.getValue(s, 'validation').toString().strstr('email') = true]

<p>
    <label for="[att.name.replaceAll(' ', '')]">[att.name.toUpperCase()]/</label>
    <input id="[att.name.replaceAll(' ', '')]" name="[att.name.replaceAll(' ', '')]" type="text" [if
att.getValue(s, 'title').toString().size() > 0] title="[att.getValue(s, 'title').toString()]/[/if]><br>
</p>
    [comment validacion de un textImput simple /]
    [elseif att.getValue(s, 'validation').toString().strstr('simpleTextImput') = true]

<p>
    <label for="[att.name.replaceAll(' ', '')]">[att.name.toUpperCase()]/</label>
    <input id="[att.name.replaceAll(' ', '')]" name="[att.name.replaceAll(' ', '')]" type="text" [if
att.getValue(s, 'title').toString().size() > 0] title="[att.getValue(s, 'title').toString()]/[/if]><br>
</p>
    [comment validacion del tipo agree checkbox /]
    [elseif att.getValue(s, 'validation').toString().strstr('agreeCheck') = true]

<p>
    <label for="[att.name.replaceAll(' ', '')]">[att.name.toUpperCase()]/</label>
    <input type="checkbox" id="[att.name.replaceAll(' ', '')]" name="[att.name.replaceAll(' ', '')]">
</p>

[comment validacion del tipo radio choice/]
[elseif att.getValue(s, 'validation').toString().strstr('radioChoice') = true]

<fieldset>
    <legend>[att.name.replaceAll(' ', '')].toUpperCase()</legend>
        [for (a : String | att.getValue(s, 'choiceOptions').toString().tokenize(',')) ]
        <label for="[" + a + "Required"]">[a.replace(' Required', '')]/[else][a.toString().replaceAll(' ','')]"/[/if]>
            <input type="radio" id="[" + a + "Required"]">[a.replace(' Required', '')]/[else][a]/[/if]
name="[" + att.name.replaceAll(' ', '')]" value="[" + a + "Required"]">[a.replace(' Required', '')]/[else][a]/[/if] [if
a.strstr('Required')]>[if a.strstr('Required')][a.toUpperCase().replace(' Required', '')]/[else][a.toUpperCase()]/[/if]
</label>
        [/for]
</fieldset>
    [if]
    [/if] [comment fin del 'richFieldLiveValidation' /]
    [if s.name.toString() = 'textInput']
        <label for="[" + att.name.replaceAll(' ', '')]">[att.name.toUpperCase()]/</label><input id="[" + att.name.replaceAll(' ','')] name="[" + att.name.replaceAll(' ', '')]" type="text" [if att.getValue(s, 'title').toString()]">[if att.getValue(s, 'password') = false] type="text" [else] type="password" [if att.getValue(s, 'mandatory') = true] required[/if]><br>
        [if]
        [if s.name.toString() = 'htmlText']
            <att.getValue(s, 'html')/">
                [if] [comment fin del htmlText' /]
                [if s.name.toString() = 'multimedia']
                    [if att.getValue(s, 'type').toString().strstr('video') = true]
                        [if att.getValue(s, 'url').toString().size() > 0]

```

```

<source src="[att.getValue(s, 'path')]" [if att.getValue(s, 'path').toString().strchr('.mp3') = true]
type="audio/mpeg">
[else] type="audio/ogg"> [/if]
Your browser does not support the audio element.
</audio>
[/if]

[if]
[if] [comment fin del multimedia /]
[if s.name.toString() = 'externalLink']

<a href="[att.getValue(s, 'url')]" [if att.getValue(s, 'title').toString().size() <> 0] title="[att.getValue(s, 'title')]">[att.name.toUpperCase()]/</a>
[if] [comment fin del externalLink /]
[if s.name.toString() = 'anchor']
<a href="[att.getValue(s, 'virtualStateName').toString().toUpperCase().concat('.php')]" [if att.getValue(s, 'title').toString().size() <> 0] title="[att.getValue(s, 'title')]">[if]>[att.name.toUpperCase()]/</a>
[if] [comment fin del anchor /]
[if s.name.toString() = 'text']
<p[if att.getValue(s, 'title').toString().size() <> 0] title="[att.getValue(s, 'title')]">[if]>[att.getValue(s, 'localText')]/</p>
[if]
[if s.name.toString() = 'button']
<p> <input type="submit" value="[att.name.toUpperCase()]">
</p>
[if]
[if s.name.toString() = 'list']
[if att.getValue(s, 'listType').toString().strchr('choice') = true ]
[if att.getValue(s, 'filter').toString().size() <> 0 ]

<fieldset>
<legend>[att.name]/</legend>

[for (a : String | att.getValue(s, 'filter').toString().tokenize(',') ) ]
<input type="radio" id="[a]" value="[a]" name="[att.name]"> [a.toUpperCase()]/
[for]
</fieldset>
[if] <br>
[if]
[if att.getValue(s, 'listType').toString().strchr('dropBox') = true ]
[if att.getValue(s, 'filter').toString().size() <> 0 ]
<p>[att.getValue(s, 'title')]/<input list="[att.getValue(s, 'title').toString().replaceAll(' ', '')]">
name="[att.getValue(s, 'title').toString().replaceAll(' ', '')]">
<datalist id="[att.getValue(s, 'title').toString().replaceAll(' ', '')]">

[for (a : String | att.getValue(s, 'filter').toString().tokenize(',') ) ]
<option value="[a]">
[for]
</datalist>
<p>
[if] <br>
[if]
[if]
[for]
[if]
</form>
[elseif c3.hasStereotype('compositeUIElement') and c3.hasStereotype('form')=false and c3.getValue(s1, 'content').toString().strchr('withinRichTab')]
[for (att: Property | c3.getAllAttributes()->asSequence())]
[for (s: Stereotype | att.getAppliedStereotypes()->asSequence())]
[if s.name.toString() = 'htmlText']
[att.getValue(s, 'html')]/
[if]
[if s.name.toString() = 'multimedia']
[if att.getValue(s, 'type').toString().strchr('video') = true ]
[if att.getValue(s, 'url').toString().size() <> 0 ]

<object width="420" height="315"
 data="[att.getValue(s, 'url')]">
</object>
[if]
[if att.getValue(s, 'path').toString().size() <> 0]
<video width="320" height="240" controls>
<source src="[att.getValue(s, 'path')]" [if att.getValue(s, 'path').toString().strchr('.mp4') = true] type="video/mp4">
[else] type="video/ogg"> [/if]
Your browser does not support the video tag.
</video>
[if]
[if]
[if att.getValue(s, 'type').toString().strchr('image') = true ]
[if att.getValue(s, 'url').toString().size() <> 0 ]


[if]
[if att.getValue(s, 'path').toString().size() <> 0]

[if]
```

```

        [if att.getValue(s, 'type').toString().strchr('sound') = true ]
<audio controls>
    <source src="[att.getValue(s, 'path').toString()]" [if att.getValue(s, 'path').toString().size() <> 0]
            [else] type="audio/ogg"> [/if]
            Your browser does not support the audio element.
</audio>
        [/if]
        [/if]
        [/if] [comment fin del 'multimedia' /]
        [if s.name.toString() = 'externalLink']
<a href="[att.getValue(s, 'url')]" [if att.getValue(s, 'title').toString().size() <> 0 ] title="[att.getValue(s, 'title')]">
[/if]>[att.name.toUpperCase()]/</a>
        [/if]
        [if s.name.toString() = 'anchor']
<a href="[att.getValue(s, 'virtualStateName').toString().toUpperCase().concat('.php')]" [if att.getValue(s, 'title').toString().size() <> 0 ] title="[att.getValue(s, 'title')]" [/if]>[att.name.toUpperCase()]/</a>
        [/if] [comment fin del anchor /]
        [if s.name.toString() = 'text']
<p[if att.getValue(s, 'title').toString().size() <> 0 ] title="[att.getValue(s, 'title')]" [/if]>[att.getValue(s, 'localText')]/</p>
        [/if]
        [if s.name.toString() = 'button']
<p>
    <input type="submit" value="[att.name.toUpperCase()]">
</p>
        [/if]
        [if s.name.toString() = 'list']
            [if att.getValue(s, 'listType').toString().strchr('choice') = true ]
            [if att.getValue(s, 'filter').toString().size() <> 0 ]
<fieldset>
<legend>[att.name]/</legend>

        [for (a : String | att.getValue(s, 'filter').toString().tokenize(',') ) ]
<input type="radio" id="[a]" value="[a]" name="[att.name]"> [a.toUpperCase()]/
        [/for]
</fieldset>
        [/if] <br>
        [/if]
        [if att.getValue(s, 'listType').toString().strchr('dropBox') = true ]
            [if att.getValue(s, 'filter').toString().size() <> 0 ]
<p>[att.getValue(s, 'title')]/<input list="[att.getValue(s, 'title').toString().replaceAll(' ', '')]">
name="[att.getValue(s, 'title').toString().replaceAll(' ', '')]">
<datalist id="[att.getValue(s, 'title').toString().replaceAll(' ', '')]">

        [for (a : String | att.getValue(s, 'filter').toString().tokenize(',') ) ]
            <option value="[a]">
        [/for]
        </datalist>
<p>
        [/if] <br>
        [/if]
        [/for]
        [/for]
        [/if]
        [if c3.getValue(s1, 'content').toString().strchr('withinARichTab')]
</div>
        [/if]
        [/for]
        [/for]
</div>
        [/if]
        [/for]
        [/for]
        [/comment en caso que sea una tabla/]
        [if (c.hasStereotype('table'))]
        [/if]
[for (c2: Class | c.followingSiblings(Class))]
[for (s1: Stereotype | c2.getAppliedStereotypes())]
    [if c2.hasStereotype('compositeIElement') and c2.hasStereotype('form')=false and c2.hasStereotype('richTabs')=false and
c2.hasStereotype('richAccordion')=false and c2.getValue(s1, 'content').toString().strchr('outsideAPanel')]
<div id="[c2.name.replaceAll(' ', '')]">
    [for (att: Property | c2.getAllAttributes())]
        [for (s: Stereotype | att.getAppliedStereotypes())]
            [if s.name.toString() = 'multimedia']
                [if att.getValue(s, 'type').toString().strchr('video') = true ]
                [if att.getValue(s, 'url').toString().size() <> 0 ]

<object width="420" height="315"
    data="[att.getValue(s, 'url')]">
</object>
        [/if]
        [if att.getValue(s, 'path').toString().size() <> 0]
<video width="320" height="240" controls>
    <source src="[att.getValue(s, 'path')]" [if att.getValue(s, 'path').toString().strchr('.mp4') = true ] type="video/mp4">
            [else] type="video/ogg"> [/if]
            Your browser does not support the video tag.
</video>
        [/if]

```

```

[/if]
    [if att.getValue(s, 'type').toString().strstr('image') = true ]
        [if att.getValue(s, 'url').toString().size() <> 0 ]

<
    [/if]
        [if att.getValue(s, 'path').toString().size() <> 0]
            [if att.getValue(s, 'path').toString().size() <> 0]


    [/if]
        [if att.getValue(s, 'type').toString().strstr('sound') = true ]
            [if att.getValue(s, 'path').toString().size() <> 0]

<audio controls>
    <source src="[att.getValue(s, 'path')/]" [if att.getValue(s, 'path').toString().strstr('.mp3') = true] type="audio/mpeg">
        [else] type="audio/ogg"> [/if]
            Your browser does not support the audio element.
    </audio>
    [/if]
        [/if]
            [comment fin del 'multimedia' /]
                [if s.name.toString() = 'externalLink']
                    <a href="[att.getValue(s, 'url')/]" [if att.getValue(s, 'title').toString().size() <> 0] title= "[att.getValue(s, 'title')/]" [/if]>[att.name.toUpperCaseFirst()]/</a>
                    [/if]
                        [if s.name.toString() = 'anchor']
                            <a href="[att.getValue(s, 'virtualStateName').toString().toUpperCaseFirst().concat('.php')/]" [if att.getValue(s, 'title').toString().size() <> 0] title= "[att.getValue(s, 'title')/]" [/if]>[att.name.toUpperCaseFirst()]/</a>
                            [/if] [comment fin del anchor /]
                                [if s.name.toString() = 'htmlText']
                                    [att.getValue(s, 'html')/]
                                    [/if]
                                        [if s.name.toString() = 'text']
                                            <p[if att.getValue(s, 'title').toString().size() <> 0] title= "[att.getValue(s, 'title')/]" [/if]>[att.getValue(s, 'localText')/]</p>
                                            [/if]
                                                [if s.name.toString() = 'list']
                                                    [if att.getValue(s, 'listType').toString().strstr('choice') = true ]
                                                        [if att.getValue(s, 'filter').toString().size() <> 0]
                                                <fieldset>
                                                    <legend>[att.name/]</legend>
[/if]
    [for (a : String | att.getValue(s, 'filter').toString().tokenize(',') ) ]
        <input type="radio" id="[a/]" value="[a/]" name="[att.name/]"/> [a.toUpperCaseFirst()/
        [/for]
    </fieldset>
    [/if] <br>
        [/if]
            [if att.getValue(s, 'listType').toString().strstr('dropBox') = true ]
                [if att.getValue(s, 'filter').toString().size() <> 0]
                    <p>[att.getValue(s, 'title')/]<input list="[att.getValue(s, 'title').toString().substituteAll(' ', '')/]" name="["[att.getValue(s, 'title').toString().substituteAll(' ', '')/]">
                        <datalist id="["[att.getValue(s, 'title').toString().substituteAll(' ', '')/]">
[/for (a : String | att.getValue(s, 'filter').toString().tokenize(',') ) ]
    <option value="["[a/]">
    [/for]
        </datalist>
    <p>
        [/if] <br>
            [/if]
                [if s.name.toString() = 'button']
                    <p>
                        <input type="submit" value="["[att.name.toUpperCaseFirst()/">
                    </p>
                    [/if]
                [/for]
            [/for]
        </div>
        [elseif s1.name.toString() = 'form' and c2.getValue(s1, 'content').toString().strstr('outsideAPanel')]
        <form id="["[c2.name.substituteAll(' ', '')/]" method="POST" action="">
            <fieldset>
                [for (att: Property | c2.getAllAttributes()->asSequence())]
                    [for (s: Stereotype | att.getAppliedStereotypes()->asSequence())]
                        [if s.name.toString() = 'richAutoSuggest']

                <p>
                    <label for="["[att.name.substituteAll(' ', '')/]">[att.name.toUpperCaseFirst()]/</label>
                    <input id="["[att.name.substituteAll(' ', '')/]" type="text" name="["[att.name.substituteAll(' ', '')/]">
                </p>
                [/if] [comment fin del richAutoSuggest /
                    [if s.name.toString() = 'richDatePicker']

                <p>[att.name.toUpperCaseFirst()/] <input type="text" id="["[att.name.substituteAll(' ', '')/]" name="["[att.name.substituteAll(' ', '')/]">[att.getValue(s, 'title').toString()/">[/if]></p>
                    [/if] [comment fin del richDatePicker /
                        [if s.name.toString() = 'richFieldLiveValidation']
                            [comment validacion del tipo password /

```

```

[if att.getValue(s, 'validation').toString().strstr('password') = true ]

<p>
    <label for="[att.name.substringAll(' ', '')/]">[att.name.toUpperCase()]/</label>
    <input id="[att.name.substringAll(' ', '')/]" name="[att.name.substringAll(' ', '')/]" type="password" [if
att.getValue(s, 'title').toString().size() > 0] title="[att.getValue(s, 'title').toString()]/"[/if]><br>
</p>
    [comment si el tagged value confirmPass esta definido /]
    [if att.getValue(s, 'confirmPass') = true]

<p>
    <label for="confirm_[att.name]">Confirm [att.name.toUpperCase()]/</label>
    <input id="confirm_[att.name.substringAll(' ', '')/]" name="[att.name.substringAll(' ', '')/]" type="password"
type="password"
</p>
    [/if]
    [comment validacion del tipo email /]
    [elseif att.getValue(s, 'validation').toString().strstr('email') = true]

<p>
    <label for="[att.name.substringAll(' ', '')/]">[att.name.toUpperCase()]/</label>
    <input id="[att.name.substringAll(' ', '')/]" name="[att.name.substringAll(' ', '')/]" type="email" [if
att.getValue(s, 'title').toString().size() > 0] title="[att.getValue(s, 'title').toString()]/"[/if]><br>
</p>
    [comment validacion de un textImput simple /]
    [elseif att.getValue(s, 'validation').toString().strstr('simpleTextImput') = true]

<p>
    <label for="[att.name.substringAll(' ', '')/]">[att.name.toUpperCase()]/</label>
    <input id="[att.name.substringAll(' ', '')/]" name="[att.name.substringAll(' ', '')/]" type="text" [if
att.getValue(s, 'title').toString().size() > 0] title="[att.getValue(s, 'title').toString()]/"[/if]><br>
</p>
    [comment validacion del tipo agree checkbox /]
    [elseif att.getValue(s, 'validation').toString().strstr('agreeCheck') = true]

<p>
    <label for="[att.name.substringAll(' ', '')/]">[att.name.toUpperCase()]/</label>
    <input type="checkbox" id="[att.name.substringAll(' ', '')/]" name="[att.name.substringAll(' ', '')/]">
</p>
    [comment validacion del tipo radio choice/]
    [elseif att.getValue(s, 'validation').toString().strstr('radioChoice') = true]

<fieldset>
    <legend>[att.name.substringAll(' ', '')].toUpperCase()</legend>
        [for (a : String | att.getValue(s, 'choiceOptions').toString().tokenize(',')) ]
        <label for="[" + a + ".strStr('Required')]">[a.replace(' Required', '')/][else][a.toString().substringAll(' ', '')/]"[/if]>
            <input type="radio" id="[" + a + ".strStr('Required')]">[a.replace(' Required', '')/][else][a/]"[/if]
name="[" + a + ".substringAll(' ', '')/]" value="[" + a + ".strStr('Required')]">[a.replace(' Required', '')/][else][a/]"[/if] [if
a.strStr('Required')]"required[/if]>[if a.strStr('Required')]"[a.toUpperCase().replace(' Required', '')/][else][a.toUpperCase()]"[/if]
            </label>
        [/for]
</fieldset>
    [/if]
    [/if] [comment fin del 'richFieldLiveValidation' /]
    [if s.name.toString() = 'textInput']
        <label for="[" + att.name.substringAll(' ', '')/]">[att.name.toUpperCase()]/</label><input id="[" + att.name.substringAll(' ', '')/]" name="[" + att.name.substringAll(' ', '')/]" type="text" [if att.getValue(s, 'title').toString()]"[/if][if att.getValue(s, 'password') = false] type="password" [if att.getValue(s, 'mandatory') = true] required[/if]><br>
        [/if]
        [if s.name.toString() = 'htmlText']
            <att.getValue(s, 'html')/>>
                [/if] [comment fin del htmlText' /]
                [if s.name.toString() = 'multimedia']
                    [if att.getValue(s, 'type').toString().strstr('video') = true]
                        [if att.getValue(s, 'url').toString().size() > 0]

```

```

<source src="[att.getValue(s, 'path')]" [if att.getValue(s, 'path').toString().strchr('.mp3') = true]
type="audio/mpeg">
[else] type="audio/ogg"> [/if]
Your browser does not support the audio element.
</audio>
[/if]

[if]
[comment fin del multimedia /]
[if s.name.toString() = 'externalLink']

<a href="[att.getValue(s, 'url')]" [if att.getValue(s, 'title').toString().size() <> 0 ] title="[att.getValue(s, 'title')]">
[if]>[att.name.toUpperCase()]/</a>
[if] [comment fin del externalLink /]
[if s.name.toString() = 'anchor']
<a href="[att.getValue(s, 'virtualStateName').toString().toUpperCase().concat('.php')]" [if att.getValue(s, 'title').toString().size() <> 0 ] title="[att.getValue(s, 'title')]"> [/if]>[att.name.toUpperCase()]/</a>
[if] [comment fin del anchor /]
[if s.name.toString() = 'text']
<p[if att.getValue(s, 'title').toString().size() <> 0 ] title="[att.getValue(s, 'title')]"> [/if]>[att.getValue(s, 'localText')]/</p>
[if]
[if s.name.toString() = 'button']
<p>
<input type="submit" value="[att.name.toUpperCase()]">
</p>
[if]
[if s.name.toString() = 'list']
[if att.getValue(s, 'listType').toString().strchr('choice') = true ]
[if att.getValue(s, 'filter').toString().size() <> 0 ]
<fieldset>
<legend>[att.name]/</legend>

[for (a : String | att.getValue(s, 'filter').toString().tokenize(',') ) ]
<input type="radio" id="[a]" value="[a]" name="[att.name]"> [a.toUpperCase()]
[for]
</fieldset>
[if] <br>
[if]
[if att.getValue(s, 'listType').toString().strchr('dropBox') = true ]
[if att.getValue(s, 'filter').toString().size() <> 0 ]
<p>[att.getValue(s, 'title')]/<input list="[att.getValue(s, 'title').toString().replaceAll(' ', '')]">
name="[att.getValue(s, 'title').toString().replaceAll(' ', '')]">
<datalist id="[att.getValue(s, 'title').toString().replaceAll(' ', '')]">

[for (a : String | att.getValue(s, 'filter').toString().tokenize(',') ) ]
<option value="[a]">
[for]
</datalist>
<p>
[if] <br>
[if]
[if]
[for]
[if]
[for]
</fieldset>
</form>
[if]
[for]
[for]
</body>
</html>

[/file]
[if]
[/template]

```