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: Lic. Iván López

Coordinadoras: Ing. Magalí González, M.Sc. Nathalie Aquino

Asunción- 2015

1- INTRODUCCIÓN

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 dichas aplicaciones, ofreciendo mejoras substanciales con respecto a las aplicaciones Web tradicionales, con nuevas características referentes a la comunicación, la distribución de los datos y la computación en el lado cliente, acompañadas además de interfaces mucho más interactivas, en las que el usuario, en ocasiones, no distingue si está utilizando la aplicación *online* u *offline*. Con estos avances propuestos por las *RIA*, muchas de las metodologías Web tradicionales basadas en la Web 1.0, tales como *WebML*[15], *UWE*[9], *OOH*[17], *OOHDM*[13] y *OOWS*[2], 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 [6] [7] [10] [4].

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, a partir de un análisis, el hecho de que sólo *OOH4RIA*, *UWE-R* y Patrones *RIA* con *UWE*, poseen tales características. Sin embargo, en estas metodologías, 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. Un nuevo enfoque para el desarrollo de aplicaciones Web basado en modelos y fundamentado en los principios propuestos por la *OMG*[1], se ha propuesto en el *DEI*[2]. Este enfoque está basado en los estándares *MDA*[3] y ofrece un esquema de modelado en capas para la separación de conceptos. Dicho enfoque se denomina *MoWebA* [12][11], y en la actualidad cuenta con características de modelado a nivel de presentación, lógica de negocio, navegación y adaptabilidad de los usuarios, pudiendo generarse aplicaciones Web completas y funcionales con modelos independientes de la plataforma. *MoWebA* tiene la capacidad de llevar a cabo

[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

extensiones a sus metamodelos para cubrir nuevas características, lo cual la hace adaptable a los cambios actuales.

El objetivo de este trabajo de fin de carrera se enmarca en la idea de llevar a cabo extensiones a la propuesta Web *MoWebA* con respecto a la capa de *Presentación*, con el fin de abarcar a algunas de las principales características de las *RIA*. Para llevar a cabo esta propuesta de extensión, se divide el esquema de trabajo de la siguiente manera. Se definen primeramente las *RIA*, presentando sus principales características y los nuevos aportes a las aplicaciones Web tradicionales. Posteriormente, se presenta el estado del arte de las metodologías de desarrollo basadas en modelos *MDD* que dan cobertura a características de *RIA*. A continuación se presenta la aproximación de desarrollo Web *MoWebA*. Se extiende luego el metamodelo 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. Como siguiente paso se evalúan los metamodelos de Contenido *y* Estructura extendidos con una ilustración. Se finaliza el trabajo con un análisis de los resultados obtenidos, elaborando la conclusión y los posibles trabajos futuros.

2-RICH INTERNET APPLICATIONS (*RIA*)

El término *RIA* fue introducido en marzo de 2002 por la empresa Macromedia (actualmente Adobe) que en ese entonces abordaba las limitaciones en cuanto a la riqueza de las interfaces, medios y contenidos de las aplicaciones [5]. En la Figura 1 se presentan los tres aspectos que, en conjunción, dieron origen a las *RIA*. Las *RIA* son aplicaciones Web que exhiben *widgets*, comportamientos y características que están presentes en las aplicaciones de escritorio. 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: a) el paradigma de página única; b) un avanzado esquema de comunicación, con la inclusión de tecnologías *push,* comunicación asíncrona entre el cliente y el servidor, y un manejo optimizado de los datos, reduciendo las solicitudes al servidor; y c) la inclusión de un motor en el cliente, en la forma de máquina virtual o extensiones (*plug-ins*) en el navegador, que administra la disposición gráfica de los elementos y la mayoría de las interacciones locales [3].

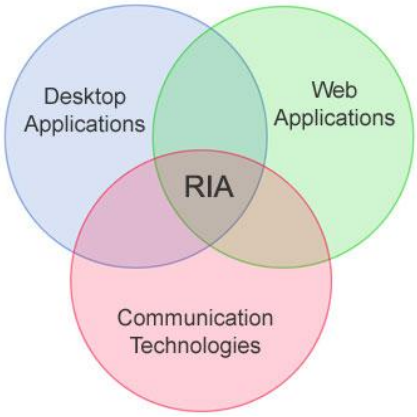

Figura 1 Fundamento de las *RIA*. Las aplicaciones Web, las aplicaciones de escritorio y las tecnologías de comunicación.

2.1 Características principales de las *RIA*

A continuación, se presentan más detalles de las cuatro características principales de las aplicaciones *RIA*.

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.

[4] **jQuery UI 1.11 API Documentation:** http://api.jqueryui.com/ 2015
[5] **jQuery Validation Plugin:** http://jqueryvalidation.org/ 2015

Lógica de negocio: en las *RIA* es posible llevar a cabo operaciones complejas directamente en el cliente (por ejemplo: efectuar navegaciones, realizar filtrados y ordenamiento de los datos con múltiples criterios, operaciones de dominio específico para sistemas complejos, validación local de datos, etc.). 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 [15].

Comunicación entre el cliente y el servidor: 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.

Presentaciones enriquecidas: las interfaces de usuario ofrecen una mayor riqueza con el manejo de eventos en el lado del cliente y los *widgets* interactivos, que son microprogramas empotrados dentro de las páginas Web y administradas por un motor de *widgets* (que podría ser un *plug-in* instalado en el navegador). Los *widgets* son microprogramas que cumplen una función predeterminada y cuyas propiedades pueden ser modificadas para expresar comportamientos personalizados por parte del usuario. Una vez modificadas las propiedades del *widget*, éste es introducido dentro de la aplicación para cumplir una función en particular. Los elementos multimedia dentro de las páginas, como la intrusión de audio y video de alta calidad, a la par de las animaciones, también son características típicas de las *RIA*, como así también, la capacidad de arrastrar y soltar elementos dentro de la interfaz, las auto-sugerencias de datos a medida que se va escribiendo un patrón en un campo, y el refrescado automático de las páginas (o porciones de esta).

2.2 Herramientas para el desarrollo de las *RIA*

Existen diferentes tecnologías para el desarrollo e implementación de las *RIA*. Las implementaciones basadas en *Javascript* o librerías *Ajax* son las más utilizadas en la actualidad, debido a que utilizan tecnologías de uso abierto y estandarizado como *Javascript*, HTML y *CSS*. 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. Entre los principales *frameworks* de desarrollo para las *RIA*[6] se pueden citar a *jQueryUI, JQuery Validation, Prototype, MooTools, YUI Library, Dojo Toolkit, Angular.JS,* entre otros.

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

3-PRINCIPALES ENFOQUES DE DESARROLLO WEB BASADO EN MODELOS PARA LAS *RIA*

[6] **List of Ajax frameworks :** https://en.wikipedia.org/wiki/List_of_Ajax_frameworks 2015

En [7] y [6] 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 [4]. Las metodologías tenidas en cuenta en el análisis que se ha realizado en este trabajo son las que contribuyen a la investigación que proviene de la comunidad de ingeniería Web, y derivan de la evolución de los enfoques dirigidos por modelos, concebidos originalmente para el diseño y desarrollo de aplicaciones Web tradicionales. Dichas metodologías son las siguientes: *WebML-RIA*[15], *OOHDM-RIA*[13], *OOH4RIA*[17], *UWE-R*[9], Patrones *RIA* con *UWE*[14] y *UWE+RUX* [8]).

De todas las metodologías anteriormente mencionadas, ninguna ofrece cobertura completa a todas las características de las *RIA* previamente presentadas, he allí la necesidad de proponer extensiones a tales metodologías o bien promover nuevas. Si se desea extender alguna metodología, una característica deseable es que el lenguaje de modelado que se utilice sea estándar y moderado por una comunidad (como *UML*). Otra característica importante es que sus modelos *PIM* (*Platform Independent Model*), no contengan detalles de una plataforma destino. Tan solo *OOH4RIA* y Patrones *RIA* con *UWE* poseen un lenguaje de modelado cien por ciento basado en *UML*, pero sus modelos *PIM* poseen detalles de alguna arquitectura destino. Es debido a este hecho particular que se optó por una nueva metodología para contemplar características *RIA,* y la aproximación *MoWebA* aparece como una opción interesante para ser extendida.

4-LA APROXIMACIÓN *MOWEBA* (MODEL ORIENTED WEB APPROACH)

MoWebA [11][12] es una propuesta creada en el *DEI* que adopta los principios de *MDA*. *MoWebA* consta de fases, niveles y aspectos. 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 de modelado son las siguientes:

a. **Modelado del problema:** en la que se incluyen al *CIM* (*Computation Independent Model*) y al *PIM* (*Platform Independent Model*). El *CIM* está orientado al modelado de los requisitos funcionales. El *PIM* está 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.

b. **Modelado de la solución**: en donde forman parte el *ASM* (*Architectural Specific Model*) y el *PSM* (*Platform Specific Model*). Es en esta fase en la que 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 la aproximación sea bastante prometedora para la implementación de las *RIA*, debido a que existen numerosas plataformas destino para desplegarlas. En las aproximaciones que se han estudiado en el estado del arte, por lo general las extensiones *RIA* son definidas en el marco de los modelos conceptuales (*PIM*), haciendo que los modelos que deberían ser independientes de la solución, adquieran elementos que ya son propios de una arquitectura específica.

c. **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.

Cabe mencionar que los modelos de la aplicación pueden refinarse, dado que todas las fases son iterativas e incrementales. En *MoWebA* se plantea tener siempre el mismo *PIM*, y a partir de este adoptar la arquitectura correspondiente.

5- *MOWEBA* EXTENDIDO PARA INCLUIR *RIA*

La propuesta de extensión para proveer de características *RIA* a *MoWebA* a nivel de la capa de Presentación (compuesta por los metamodelos de Contenido y Estructura), consiste en agregar nuevos elementos enriquecidos (*widgets*) y propiedades de validación local de campos de entrada al metamodelo existente de Contenido (*Content*). También se agrega al metamodelo de Estructura (*Layout*) original, características para establecer el tipo de coordenadas (en pixeles o porcentajes) para el posicionamiento dentro de las páginas de cada uno de los elementos definidos en el metamodelo de Contenido. Seguidamente, para el modelado de la sintaxis concreta de las extensiones RIA agregadas a los metamodelos de la capa de Presentación, las extensiones *RIA* son introducidas dentro de los perfiles (*profiles UML*) de Contenido y Estructura con los que cuenta MoWebA. Los perfiles para la capa de Presentación con las extensiones *RIA*, permiten modelar los *PIM* para una aplicación *RIA* en particular. Posteriormente los *PIM* para la capa de Presentación de una aplicación modelada con *MoWebA,* son transformados a código para las plataformas *jQueryUI* y *jQuery Validate*. Estas transformaciones son de modelo a texto (*M2T*) y son llevadas a cabo por las plantillas (*templates*) de transformación de presentación y estructura respectivamente, que han sido implementadas utilizando la herramienta *Acceleo*.

5.1 Extensiones *RIA* a los metamodelos de Contenido y Estructura de MoWebA

Las nuevas extensiones propuestas a los metamodelos de Contenido (*Content*) y Estructura (*Layout*) de *MoWebA* se presentan en la Figura 2. En ellos se despliegan los diversos elementos que permiten representar una interfaz de usuario enriquecida. Los diferentes elementos tanto del metamodelo de Contenido como del 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.

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 [1]. Para el caso de *MoWebA*, cada *CompositeUIElement* puede contener uno o más elementos *PD Element* que a la vez pueden ser compuestos (*CompositeUIElemet*) o simples u hojas (*UIElement*). El *PD Element,* que corresponde a una clase padre abstracta, contiene las propiedades *horizontalOrder* y *verticalOrder* para indicar una prioridad en el orden de despliegue en pantalla horizontal o vertical de un elemento simple o compuesto que se encuentra definido en una misma región que otro elemento en una misma página. 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 clasificar mejor a los elementos de interfaz y para una mayor claridad dentro del metamodelo de *Contenido*. Los distintos *UIElements* se clasifican entonces de la siguiente forma:

- ***Elementos de salida* (*OutputElement*):** 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 (*InputElement*):** 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 *textInput*, *list*, *richAutoSuggest*, *richDatePicker, password y richEmail*.
- ***Elementos de control* (*ControlElement*)*:*** 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. A su vez, 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*.

El metamodelo de *Estructura* no ha sufrido muchos cambios con respecto a su versión original. Entre las adaptaciones que se han realizado a este metamodelo se cuentan 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 *widhValue* 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 UML, para agregar las características propias de *MoWebA* y por ende hacer posible la representación de la sintaxis concreta de *MoWebA*. Por motivos de espacio estos perfiles no se presentan en este resumen. A continuación se describen cada uno de los elementos *RIA* que forman parte de la extensión al metamodelo de *Contenido* de *MoWebA* con sus respectivas propiedades.

5.1.1 RichAutoSuggest

Este elemento de interfaz enriquecido de entrada, contiene al atributo *source*. Este atributo tiene una doble funcionalidad. Una de ellas es permitir definir 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 un campo *País de origen* del tipo *RichAutoSuggest*, el atributo *source* puede definirse como source=”Paraguay@Portugal@PaquistanPolonia@Peru@España@...”.

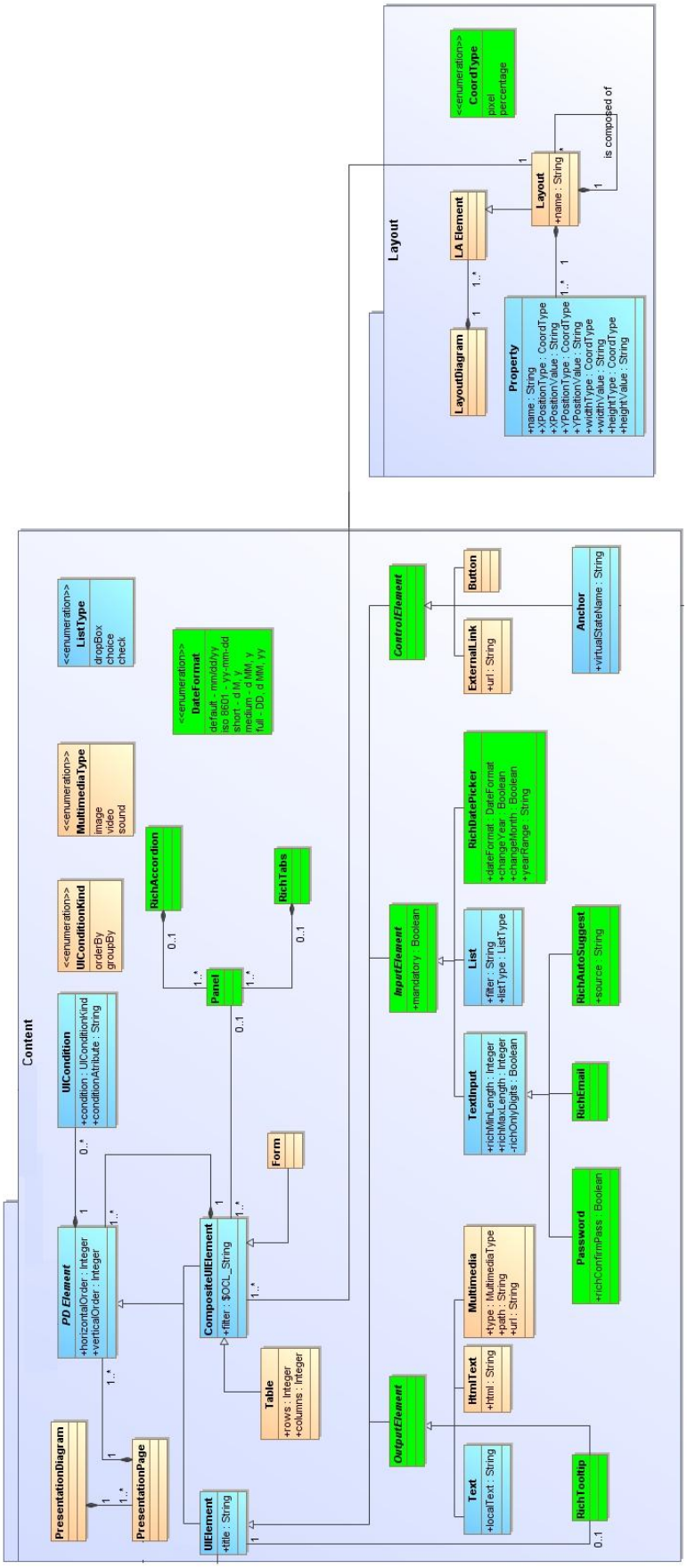

Figura 2 Metamodelo de contenido y estructura para *MoWebA* con extensiones *RIA*

La otra funcionalidad del atributo *source* permite definir 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 entre el elemento *RichAutoSuggest* y un *Value Object* que contiene la información necesaria de una entidad en particular.

5.1.2 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.

5.1.3 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.
Al definirse este elemento en conjunción con alguno 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.

5.1.4 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 sobre diversos elementos de entrada, como los del tipo *TextInput, RichEmail* y *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*, el activar el atributo *mandatory* implica que al menos una de las opciones de un *dropBox*, *choice* o *check* debe ser seleccionada.

5.1.5 RichAccordion y RichTabs

Estos *widgets* permiten encapsular a varios elementos de interfaz de *MoWebA* dentro de paneles colapsables para presentar información en una cantidad limitada de espacio. Entre los elementos que pueden ser desplegados en los paneles se encuentran los *UIElement*, en cualquiera de sus extensiones *InputElement*, *OutputElement* o *ControElement*, como así también los *CompositeUIElements*, *Table* y los *Form*.

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

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. Los *PIM* de la aplicación son modelados utilizando la herramienta *MagicDraw*[7]. Posteriormente tanto los *PIM* como los perfiles de *MoWebA* son exportados al formato *XMI* del *EMF*[8]. Esto de por sí es

[7] **No Magic:** http://www.nomagic.com/products/magicdraw.html 2015
[8] **Eclipse Modelling Framwwork:** https://www.eclipse.org/modeling/emf 2015

llevado a cabo a fines de tener compatibilidad con la herramienta de transformación *M2T Acceleo*[9], que toma como entrada modelos UML basados en el metamodelo *Ecore*[10].

Por medio de las plantillas de transformación y los módulos de servicio *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* , *.css y .js*) que representan a la aplicación en sí. En la Figura 3 se presenta el proceso de modelado y generación de interfaces enriquecidas (también conocidos como los *front-ends* de las aplicaciones). Para el caso de *MoWebA* con *RIA*, 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 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 a partir del código generado.

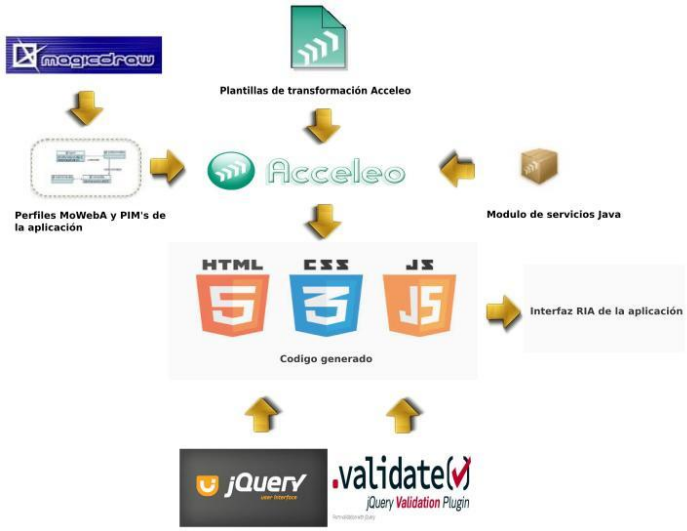

Figura 3 Fases de desarrollo para la propuesta de extensión a *MoWebA*

6-ILUSTRACIÓN DE LA PROPUESTA

Con la intensión de ilustrar de una manera práctica las extensiones llevadas a cabo a la aproximación Web *MoWebA*, se ha tomado la decisión de implementar un sistema que refleje tales extensiones. Con este trabajo, se ha logrado recabar datos que permiten intuir que la propuesta de extensión presentada ofrece cobertura a algunas de las diversas características que contemplan las *RIA* que han sido analizadas. El objetivo de esta ilustración, 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.

Si bien en una primera instancia, se ha deseado llevar a cabo un caso de estudio para validar las extensiones *RIA* hechas a la aproximación *MoWebA* y de esa forma, ofrecer una mayor formalidad a los resultados obtenidos, no fue posible implementar esa idea. Esto se debió a la principal limitante de no contar con la

[9] **Acceleo:** https://eclipse.org/acceleo 2015
[10] **Ecore:** Metamodelo nativo que forma parte del core del EMF para describir a los modelos

población debidamente instruida en la aproximación *MoWebA* para validar la extensión en el tiempo pre-estimado de desarrollo del caso de estudio. De allí, que el autor del trabajo tuvo que abocarse a la tarea de diseñar el caso, preparar la colección de datos, colectar los datos, analizar los datos colectados y reportar los resultados. Debido a que existían muchas amenazas a la validez de los datos reportados que no podían atenuarse, se optó por el método de ilustración. No obstante, con el objetivo de seguir una secuencia estructurada de pasos se ha optado por seguir las recomendaciones de Runeson [16] con respecto a cómo realizar casos de estudio.

6.1 El caso y las unidades de análisis

El caso ilustrativo consistió 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 al enfoque *MoWebA*, de una manera clara y sencilla.

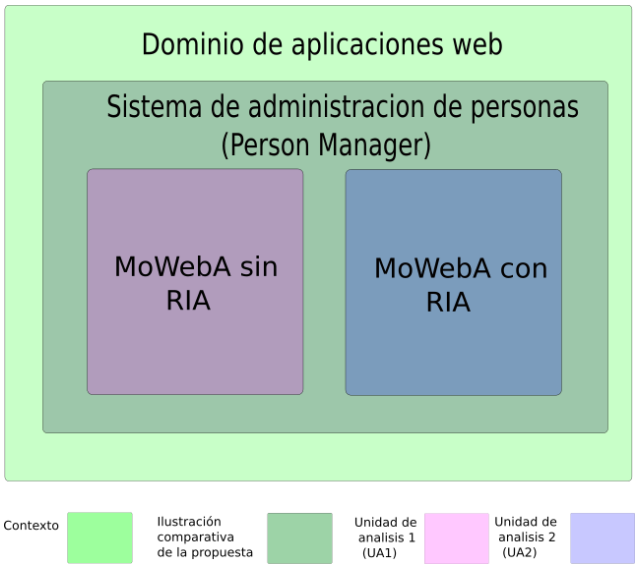

Figura 4 Ilustración del sistema *Person Manager* implementado con *MoWebA* desde dos enfoques distintos

El caso fue analizado desde dos unidades de análisis como se puede apreciar en la Figura 4. La primera unidad de análisis se refiere a la implementación de la capa de presentación del *Person Manager* con *MoWebA* sin extensiones *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*. El *Person Manager* está basado en el trabajo de Gharavi [18].

La secuencia de acciones llevadas a cabo en el transcurso de este proceso ilustrativo fue la siguiente:

a)- Se diseñó el *toy problem Person Manager* y se separó el problema en 2 unidades de análisis.
b)- Se elaboraron las preguntas de investigación de interés y se identificaron las variables de medición para la colección de los datos.
c)- Se colectaron los datos en base a las mediciones hechas.
d)- Se analizaron los datos colectados y se reportaron los resultados.

6.2 Preguntas de investigación

Las siguientes cinco preguntas de investigación (PI) fueron propuestas para el análisis:

PI1: ¿Consume 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?

6.3 Colección de los datos

Para las PI1, PI2 y PI5 los datos correspondientes a las variables de medición que fueron previamente definidas, fueron recabados y almacenados en tablas diseñadas para el efecto. Para las PI3 y PI4, las capturas de pantalla de cada uno de los enfoques implementados, fueron la fuente de datos para concluir los resultados.

6.4 Análisis e interpretación de los resultados

En base a los datos obtenidos en cada una de las mediciones, se respondió cada una de las preguntas de investigación anteriormente presentadas y se obtuvieron ciertas conclusiones:

- **Con respecto a PI1**: El enfoque *MoWebA* con *RIA* tardó 8 minutos más en el proceso de modelado que *MoWebA* sin *RIA*. Esto se debe principalmente a que para establecer características *RIA* con *MoWebA*, es necesario definir un mayor número de propiedades (valores etiquetados) intrínsecas en cada uno de los elementos que forman parte de la extensión, a diferencia de *MoWebA* sin *RIA*. En otras palabras, esto concuerda con la intuición en el sentido de que cuanto mayor es el nivel de detalle en el modelo, mayor tiempo de modelado se requiere. Sin embargo, este hecho no constituye una limitante demasiado grave, teniendo en cuenta que ese tiempo extra de modelado permite a la interfaz de la aplicación *Person Manager* enriquecerse notablemente.

- **Con respecto a PI2**: Una vez que los modelos de la aplicación se encontraban listos, el siguiente paso era generar código fuente a partir de ellos. Cuando a alguno de los modelos le faltaba definir alguna propiedad para alguno de sus elementos, entonces el código fuente generado en primera instancia, no reflejaba el resultado esperado. En estos casos, iterativamente se volvían a hacer cambios al modelo y luego se generaba de vuelta la aplicación. Un leve incremento en el número de generaciones para el enfoque *MoWebA* con *RIA* se pudo apreciar a partir de los datos recabados, con respecto a *MoWebA* sin *RIA*. Las vistas de agregar persona y de borrar persona son las que incurrieron en la mayor cantidad de generaciones de código. Debido a que los requerimientos *RIA* requieren un mayor nivel de detalle en los modelos para el caso de *MoWebA* con *RIA* con respecto a *MoWebA* sin *RIA*, existe una mayor posibilidad de cometer errores en los modelos y por ende será necesaria una mayor cantidad de generaciones de código para ir depurando la aplicación.

- **Con respecto a PI3**: El enfoque *MoWebA* con *RIA* ofrece numerosas ventajas con respecto a las presentaciones enriquecidas, evitando recargas innecesarias de las páginas y presentando *widgets* interactivos como los *richDatePicker*, *richAutoSuggest* y *richToolTip*. El enfoque *MoWebA* sin *RIA* no contempla tales elementos enriquecidos y navegar por cada una de sus páginas implica recargar completamente cada una de ellas.

- **Con respecto a PI4**: El enfoque *MoWebA* con *RIA* permite llevar a cabo diversas validaciones en los campos de entrada de la aplicación, como campos que deben ser obligatorios, longitudes mínima y máxima de caracteres en un campo, validaciones de claves y formato de email. En contraparte el enfoque *MoWebA* sin *RIA*, no contiene ningún tipo de validación.

- **Con respecto a PI5:** 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 que el enfoque con extensiones *RIA* (equivalente a un 32% menos). 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. 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 código que fue implementado manualmente en la aplicación, 53% (para el enfoque *MoWebA* sin RIA y 43% (para *MoWebA* con *RIA*) respectivamente, corresponde a código para refinar la aplicación final, y código para el acceso a la capa lógica y de dominio de la aplicación. De ahí se puede concluir que el 47% del código del *Person Manager* fue generado de manera automática para el enfoque *MoWebA* sin *RIA* y el 57% para el enfoque *MoWebA* con *RIA* .

7-CONCLUSIONES Y TRABAJOS FUTUROS

En este trabajo de fin de carrera se ha llevado a cabo un estudio detallado de las principales características y tecnologías de las *RIA* junto a una investigación del estado del arte de las principales metodologías Web basadas en *MDD* y *MDA* que ofrecen cobertura a las *RIA*. También se ha realizado un análisis de los elementos de interfaz enriquecidos (*widgets*) más utilizados. Posteriormente, se ha extendido el metamodelo de *Contenido* de *MoWebA,* agregando una nueva reestructuración y clasificación de los elementos de interfaz, separando a los distintos componentes de interfaz en elementos de entrada, salida y control, utilizando el patrón de diseño general *composite*. Seguidamente algunos *widgets interactivos* comunes en las aplicaciones *RIA* se han agregado, precisamente *richAccordion*, *richTabs*, *richAutoSuggest*, *richDatePicker* y *richToolTip* y el *live Validation.* El metamodelo de Estructura también se extendió para permitir definir cada una de las coordenadas posicionales de las páginas en pixeles o en porcentajes. Para la definición de la sintaxis concreta de la presentación, se agregaron los nuevos *widgets* al perfil de Contenido de *MoWebA,* y las nuevas coordenadas al perfil de Estructura.

Un análisis de las principales herramientas de transformación de modelo a texto (M2T) basado en plantillas también se ha llevado a cabo. Con la herramienta de transformación M2T *Acceleo,* se implementaron las plantillas de transformación para la presentación. La transformación genera código para cada uno de los elementos definidos en el perfil de *Contenido* de *MoWebA,* a partir de los *PIM* de entrada, en donde, para los *widgets* se genera código para la plataforma destino *jQueryUI* y *jQuery validation plug-in.* A partir del perfil de *Estructura,* se genera código *CSS* con las posiciones de los elementos de interfaz según fueron establecidas en el *PIM* de entrada. Finalmente, una ilustración evaluativa de la propuesta se llevó a cabo para presentar los aportes realizados a la capa de presentación de *MoWebA*.

Entre los trabajos futuros que podrían realizarse podrían citarse el validar la propuesta con un caso de estudio formal. También podrían incluirse otras características *RIA* a *MoWebA*, no sólo a nivel de la presentación, sino también en la comunicación cliente-servidor, en la lógica de negocios, a diferencia de las validaciones locales, y ofrecer cobertura de persistencia de datos en el lado del cliente. Otro trabajo futuro interesante sería realizar transformaciones para otros *frameworks* o plataformas destino *RIA*.

BIBLIOGRAFÍA

[1] Freeman E, Robson E, Sierra K, and Bates B. *Head first Design Patterns*, ISBN 978-0-5960-07126. O' Reilly Media, 2014.

[2] 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.

[3] Martínez-Ruiz F J. *A Development Method for User Interfaces of Rich Internet Applications*. PhD thesis, Université catholique de Louvain, Belgium, August 2010.

[4] 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.

[5]Allaire J. Requirements for rich internet applications. http://download.macromedia.com/pub/flash/whitepapers/richclient.pdf, March 2002.

[6] 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, 2008. Australian Computer Society, Inc.

[7] 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.

[8] Preciado J C, Linaje M, Morales-Chaparro R, Sanchez-Figueroa F, Zhang G, Kroiβ 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] Runeson P, Höst M, Rainer A, and Regnell B. *CASE STUDY RESEARCH IN SOFTWARE ENGENEERING. Guidelines and Examples*, ISBN 978-1118104354. Jhon Wiley & Sons,Inc, Hoboken, New Jersey, 2012.

[17] 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.

[18] Vahid Gharavi S V. Model-driven development of *Ajax* web applications. Master's thesis, Faculty EEMCS, Delft University of Technology, September 2008.