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.**

**Dra. Nathalíe 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 Applicantions* (RIA). Estas representan todo un desafío para la ingeniería Web, ya que las RIAs han dado un cambio radical en la manera en que se comportan, desarrollan y despliegan las aplicaciones Web, ofreciendo mejoras substanciales con respecto a las aplicaciones Web tradicionales, con nuevas características referentes a la comunicación, la distribución de los datos y la computación en el lado cliente, acompañadas de interfaces mucho mas interactivas, en donde el usuario en ocasiones, no distingue si está utilizando la aplicación *online* o *offline*. Con estos avances propuestos por las RIA, muchas de las metodologías Web tradicionales basadas en la Web 1.0, tales como; WebML[], UWE[], OOH[], OOHDM[] y OOWS[], 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 [] [] [] [].

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 del análisis, el hecho de que solo 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. Con ese propósito en la mira, un nuevo enfoque para el desarrollo de aplicaciones Web basado en modelos y fundamentado en los principios propuestos por la OMG[[1]](#footnote-1), se ha propuesto en el DEI[[2]](#footnote-2). Este enfoque está basado en los estándares MDA[[3]](#footnote-3) y ofrece un esquema de modelado en capas para la separación de conceptos. Dicho enfoque se denomina MoWebA[][], y en la actualidad cuenta con características de modelado a nivel de presentación, lógica de negocio, navegación y adaptabilidad de los usuarios, pudiendo generarse aplicaciones Web completas y funcionales con modelos independientes de la plataforma. MoWebA tiene la capacidad de llevar a cabo extensiones a sus metamodelos para cubrir nuevas características, lo cual la hace adaptable a los cambios actuales.

El objetivo de este trabajo de fin de carrera se enmarca en la idea de llevar a cabo extensiones a la 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 basada 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 de MoWebA y se presenta una propuesta de transformación de modelo a texto (M2T) para la plataforma destino *jQueryUI[[4]](#footnote-4)* y *jQuery Validation Plugin[[5]](#footnote-5)*, para cubrir algunas características RIA de las presentaciones enriquecidas y de la lógica de negocios en el lado cliente. Como siguiente paso se evalúa el metamodelo de Contenido y Estructura extendido con una ilustración . Se finaliza el trabajo con un análisis de los resultados obtenidos elaborando la conclusión y los posibles trabajos futuros.

**2-extensiones ria a los metamodelos de contenido y estructura de MoWebA**

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

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 []. 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 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 (*imputElements*):** 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*.

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 widhType 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. Los perfiles de Contenido y Estructura se presentan en el capítulo 4. 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.

**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@...”.

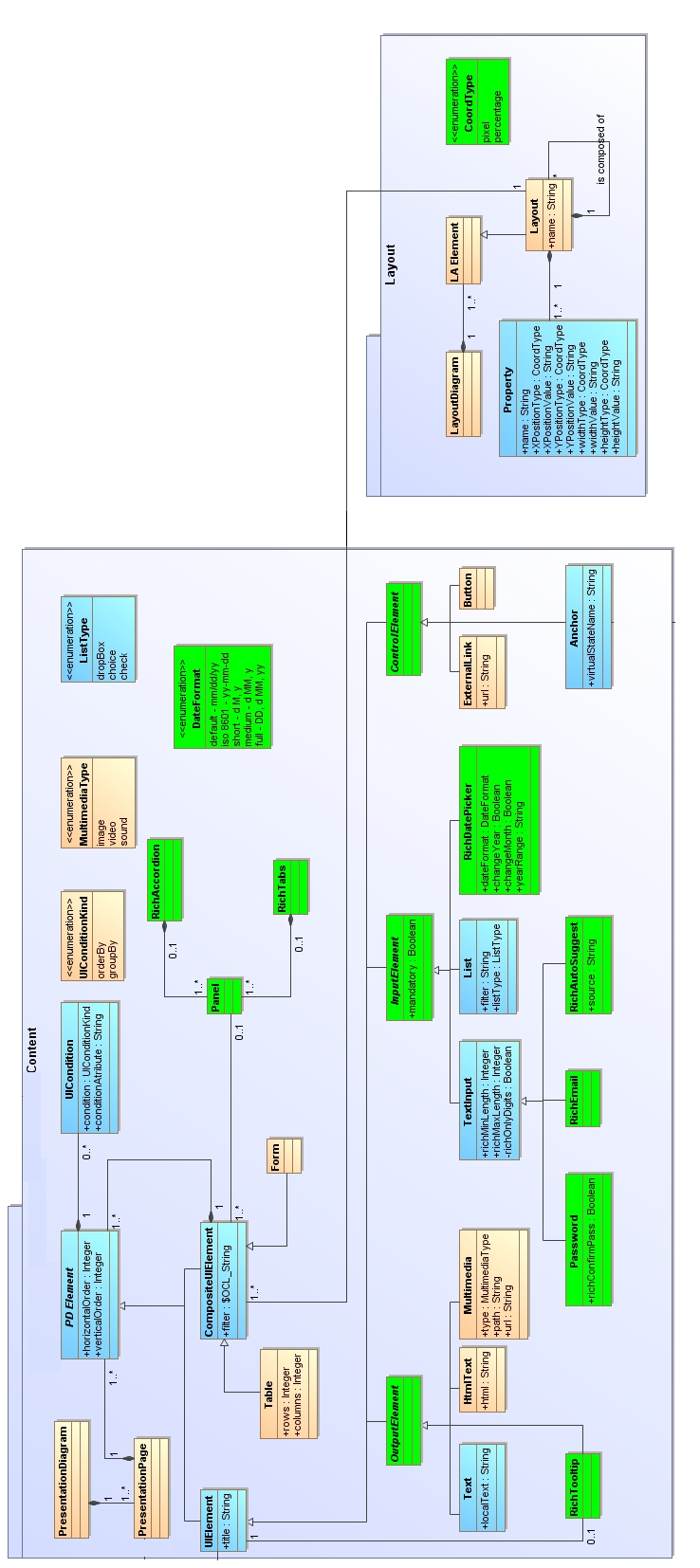


Figura 1 Metamodelo de contenido y estructura para MoWebA con extensiones RIA

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 letra d) se presenta un ejemplo del *RichAutoSuggest*.

**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 letra c) se presenta el *widget* *RichDatePicker*

**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.

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 letras b) y h) se presenta el widget *RichToolTip*.

**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,* Rich*Email*, *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 *ImputElement*, 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 letras b), e), f), g), h) e i) se presentan algunos ejemplos de *Live Validation*.

**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 *ControElement*, como así también los *CompositeUIElements*, *Table* y los *Form*. En la letra a) se presenta un ejemplo del *widget* *RichAccordion.*

**2.1.6 RichTabs**

El *RichTabs* permite al igual que el *RichAccordion* aglomerar a varios elementos de interfaz en cada una de sus pestañas o paneles como se lo denomina en MoWebA. En la 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.

**2.2- 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*[[6]](#footnote-6). Posteriormente tanto los *PIM* como los perfiles de MoWebA son exportados al formato *XMI* del *EMF*[[7]](#footnote-7). Esto de por sí es llevado a cabo a fines de tener compatibilidad con la herramienta de transformación *M2T* *Acceleo*[[8]](#footnote-8), que toma como entrada modelos UML basados en el metamodelo Ecore[[9]](#footnote-9).

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 se presenta el proceso de modelado y generación de interfaces enriquecidas (también conocidos como los *front-ends* de las aplicaciones). Para 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 2 Fases de desarrollo para la propuesta de extensión a MoWebA

**3-iLUSTRACION 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* (ver definición en el Anexo 1) 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, esta idea no fue posible implementar. Esto se debió a la principal limitante de no contar con la 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 opto 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 [].

**3.1 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.

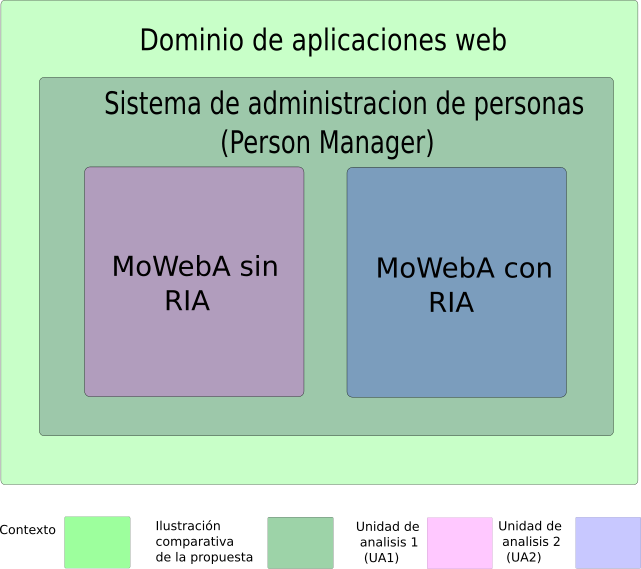


Figura 3 Ilustración del sistema *Person Manager* implementado con MoWebA desde dos enfoques distintos

El caso fue analizado desde dos unidades de Análisis como se puede apreciar en la . 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 []]

**3.2 Colección de los datos**

En esta sección se presentan los datos recabados para responder a 5 preguntas de investigación (PI). La información necesaria que se utilizó para responder a las PI1, PI2 y PI5 se encuentran en las , y . Las PI3 y PI4, fueron respondidas utilizando la y que son capturas de pantalla de la aplicación Person Manager para ambos enfoques.

|  |  |  |
| --- | --- | --- |
| **Tiempos en minutos de modelado** | **MoWebA sin RIA** | **MoWebA con RIA** |
| **Agregar persona** | 50 minutos | 56 minutos |
| **Listar persona** | 27 minutos | 28 minutos |
| **Remover persona** | 29 minutos | 30 minutos |
| **Totales** | 106 minutos | 114 minutos |

Tabla 1 Tiempo de modelado para cada uno de los enfoques

|  |  |  |
| --- | --- | --- |
| **Número de generaciones de código** | **MoWebA sin RIA** | **MoWebA con RIA** |
| **Agregar persona** | 3 | 4 |
| **Listar persona** | 1 | 1 |
| **Remover persona** | 2 | 3 |
| **Totales** | 6 | 8 |

Tabla 2 Cantidad de generaciones de código para cada uno de los enfoques para la obtención de la interfaz final

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Líneas de código** | **MoWebA sin RIA** | | | **MoWebA con RIA** | | |
| **Líneas de código automáticas** | **Líneas de código manuales** | **Totales** | **Líneas de código automáticas** | **Líneas de código manuales** | **Totales** |
| **Agregar persona** | 51 | 56 | 107 | 135 | 56 | 191 |
| **Mostrar persona** | 1 | 45 | 46 | 3 | 45 | 48 |
| **Remover persona** | 7 | 27 | 34 | 31 | 27 | 58 |
| **Estructura y código común para todas las vistas(cabecera, estructura y pié de pagina** | 67 | 10 | 77 | 52 | 38 | 90 |
| **Totales** | 126 | 138 | 264 | 221 | 166 | 387 |

Tabla 3 Líneas de código para ambos enfoques del Person Manager

**3.3 Análisis e interpretación de los resultados.**

Aquí se presentan los resultados de las mediciones realizadas en la ilustración. En base a los datos obtenidos, se respondieron a las siguientes preguntas de investigación.

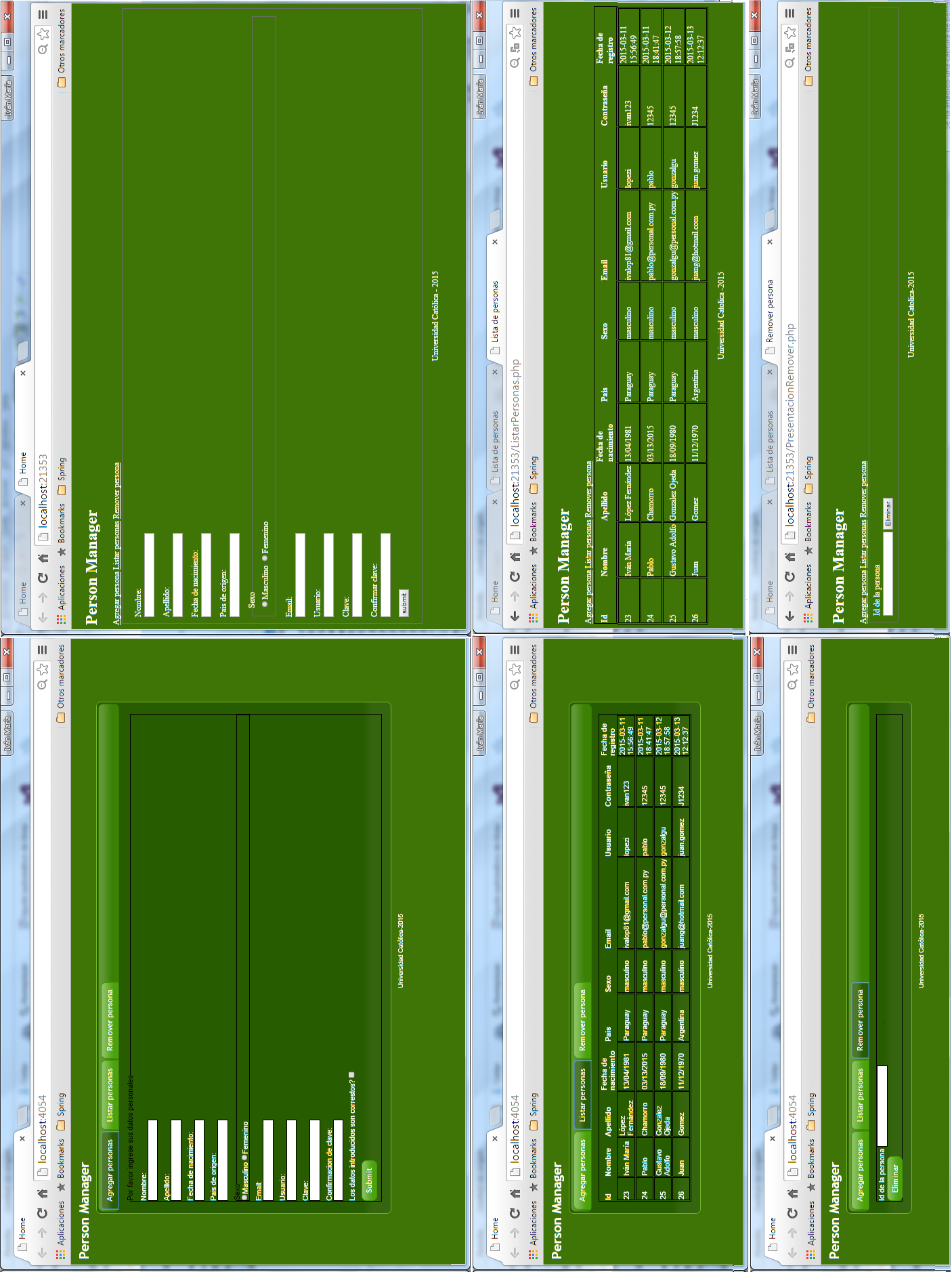


Figura 4 Vistas del *Person Manager*. Izq-MoWebA con RIA. Der MoWebA sin RIA

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

Según puede apreciarse en la , a nivel general el enfoque MoWebA con RIA deparó en 8 minutos más en el proceso de modelado que su contraparte sin RIA. El enfoque MoWebA con RIA conlleva a definir en

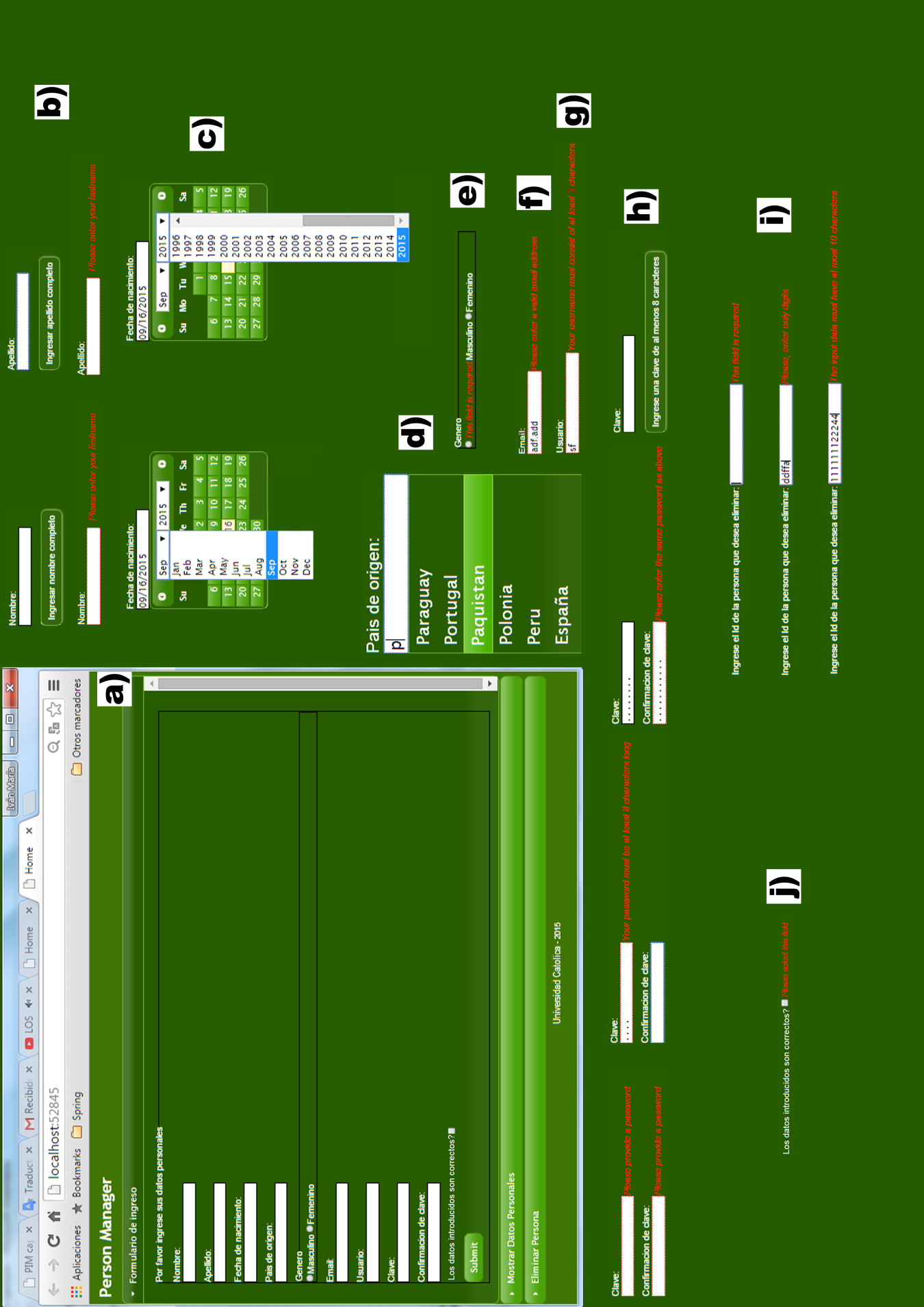
****

Figura 5 Todos los widgets del *Person Manager*

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 eso 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 conRIA hay que definir valores etiquetados para definir validaciones de campo a diferencia de MoWebA sin RIA. Desde el punto de vista práctico, que el enfoque MoWebA con RIA tarde 8 minutos más que su par sin RIA no es una limitante demasiado condicionante, teniendo en cuenta que, 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 , 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 interfazy por ende existe una mayor probabilidad de cometer fallos en el modelado, lo que incurre en una mayor cantidad de veces que la aplicación debe generarse hasta su depuración final. Para la vista Remover Persona se incurrió en un número mínimamente superior de generaciones de código implementando MoWebA con RIA, precisamente un 10% más que con el enfoque sin RIA, pudiendo deberse este hecho también, a que la vista con MoWebA con RIA contiene requerimientos de interfaz RIA a diferencia de su contraparte. 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 pueden identificarse en la y . 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***

***RichDatepicker***

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 pié a cometer errores por parte del usuario.

***RichAutoSuggest***

El Rich*AutoSuggest* posibilita 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.

***RichToolTip***

A menudo es útil complementar con información adicional los campos de entrada de los formularios Web. Con el RichToolTip 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?**

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 la 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 vacios como puede apreciarse en la letras b), e), f), g), h) , i) y j).

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 de la persona, utilizado para borrar un registro del sistema letra i). En este campo, de igual manera, no es posible ingresar más de 10 dígitos para evitar algún desbordamiento numérico. También, el campo email verifica que la cadena ingresada por el usuario corresponda a un email válido letra f).

Por último, en el campo de selección de género es mandatorio seleccionar uno de los radio controles (masculino, femenino) letra e), como así también, es mandatorio seleccionar la caja de selección del campo de conformidad letra j). 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 (derecha de la ), 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.

**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. Estos 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 cpdigo *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 MoWebaA sin RIA y 58% en su contraparte RIA).

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

**6-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 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. Seguidamente algunos *widgets interactivos* comunes en las aplicaciones RIA se han agregado, precisamente richAccordion, richTabs, richAutoSuggest, richDatePicker y richToolTip y el *live Validation.* Para la definición de la sintaxis concreta de la presentación, se agregaron los nuevos widgets al perfil de Contenido de MoWebA. Un análisis de las principales herramientas de transformación de modelo a texto (M2T) basado en plantillas. Con la herramienta de transformación (M2T) *Acceleo* se implementaron las plantillasde 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 en donde, para los *widgets* se genera código para la plataforma destino *jQueryUI* y *jQuery validation plug-in)* y de estructura (la cual genera código CSS con las posiciones 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.

Dentro de los trabajos futuros que podrían realizarse podría citarse el validar la propuesta con un caso de estudio formal, incluir otras características RIA, no solo a nivel de la presentación sino también en la lógica de negocios y ofrecer cobertura de persistencia de datos en el lado de cliente. Otro trabajo futuro interesante sería realizar transformaciones para otros frameworks o plataformas destino RIA.

**BIBLIOGRAFIA**

[1] Freeman E, Robson E, Sierra K, and Bates B. *Head first Design Patterns*, volume 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] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.

[14] Vahid Gharavi S V. Model-driven development of ajax web applications. Master’s thesis, Faculty EEMCS, Delft University of Technology, September 2008.

1. **Object Management Group:** <http://www.omg.org/> 2015 [↑](#footnote-ref-1)
2. Departamento de Ingeniería Electrónica e Informática de la Universidad Católica Nuestra Señora de la Asunción [↑](#footnote-ref-2)
3. **Model Driven Architecture:** <http://www.omg.org/mda/> 2015 [↑](#footnote-ref-3)
4. **jQuery UI 1.11 API Documentation:** <http://api.jqueryui.com/> 2015 [↑](#footnote-ref-4)
5. **jQuery Validation Plugin:** <http://jqueryvalidation.org/> 2015 [↑](#footnote-ref-5)
6. **No Magic:** <http://www.nomagic.com/products/magicdraw.html> 2015 [↑](#footnote-ref-6)
7. **Eclipse Modelling Framwwork:**  <https://www.eclipse.org/modeling/emf> 2015 [↑](#footnote-ref-7)
8. **Acceleo:** <https://eclipse.org/acceleo> 2015 [↑](#footnote-ref-8)
9. **Ecore:** Metamodelo nativo que forma parte del core del EMF para describir a los modelos [↑](#footnote-ref-9)