**Capítulo 3**

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

Se ha visto en la sección final del 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 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 WebML+ RUX o UWE+ RUX). También se ha notado que muchas de las transformaciones M2T no se llevan a cabo automáticamente sino de manera semiautomática o manual, como es el caso de UWE con patrones.

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

Primeramente se dará a conocer la sintaxis abstracta de MoWebA para el modelado de los elementos de presentación, precisamente el metamodelo de presentación, que abarca a los metamodelos de contenido (content) y estructura (layout). Luego se continúa con la presentación de los perfiles (profiles) UML para la representación de la sintaxis concreta de la presentación, tanto para el contenido como así también para el posicionamiento. Se cerrará el capítulo con algunos ejemplos de algunos PIM obtenidos a partir de los perfiles de MoWebA y una breve discusión acerca del enfoque tomado para el modelado de widgets con MoWebA.

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

La Figura 1 representa el proceso tenido en cuenta en este trabajo de fin de carrera para el modelado y generación de interfaces enriquecidas (también conocidos como los *front-ends* de las aplicaciones). 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*. Los modelos PIM y los perfiles están basados en el estándar MOF que forma parte del enfoque MDA. Los modelos PIM se crean utilizando la herramienta MagicDraw[[1]](#footnote-1). Posteriormente tanto los PIM como los perfiles son exportados al formato xmi del EMF[[2]](#footnote-2) que está basado en el metamodelo llamado Ecore. Esto de por sí es llevado a cabo a fines de tener compatibilidad con la herramienta de transformación M2T Acceleo[[3]](#footnote-3), que toma como entrada modelos UML que están basados en el lenguaje de metamodelado Ecore.

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 (.js, .html y .css) que representan a la aplicación en sí. 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 1 Fases de desarrollo para la propuesta de extensión a MoWebA

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

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

**3.2.1 - Los metamodelos de contenido y Estructura de MoWebA.**

Los metamodelos de contenido y estructura de MoWeba están representados en la Figura 2. En ellos se presentan los diversos elementos que permiten representar una interfaz de usuario. En color azul, se marcan los nuevos elementos propuestos a MoWebA, con los cuales será posible generar aplicaciones finales con los elementos enriquecidos comunes en las RIA *(widgets)* que fueron presentados en el capítulo anterior.

Los distintos elementos de interfaz de usuario se clasifican en:

* Elementos de salida (*text*, *htmlText* y *multimedia*).
* Elementos de entrada (*textInputs*, *list*, *richAutoSuggest*, *richDatePicker* y *richFieldLiveValidation*).
* Elementos de control (*externalLink*, *anchor* y *button*).

Una particularidad ocurre con el elemento *richToolTip* con respecto a la clasificación anterior. Este elemento 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, por lo tanto se encuentra definido afuera de estas clasificaciones con su respectiva relación de inclusión a cada una de ellas.

Con respecto a los elementos de interfaz compuestos (estos son elementos que pueden contener a uno o más elementos de interfaz) tenemos a los form y table, que son generalizaciones del elemento compuesto base *compositeUIElement*. Como parte de las extensiones llevadas a cabo a MoWebA, también se encuentran los Panels que contienen a los elementos *richAccordion* y los *richTabs*. Estos elementos tienen la particularidad de que pueden agrupar a muchos elementos de interfaz (*compositeUIElement*, *form*, *table*) y condensarlos en un espacio determinado de la página, que corresponde a la información desplegada al momento de seleccionar un panel para el caso del richAccordion o una pestaña para el caso de un richTab, optimizando la utilización del espacio y mejorando la interactividad con el usuario.

Posteriormente, una vez definidos los elementos de interfaz que formarán parte de la aplicación, cada uno de los *compositeUIElement* puede ser ubicado en una posición dentro la página y esta posición es parametrizada por medio de las propiedades que forman parte del metamodelo de posicionamiento (Layout).

De los metamodelos de contenido y estructua 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 en a continuación en la siguiente sección.

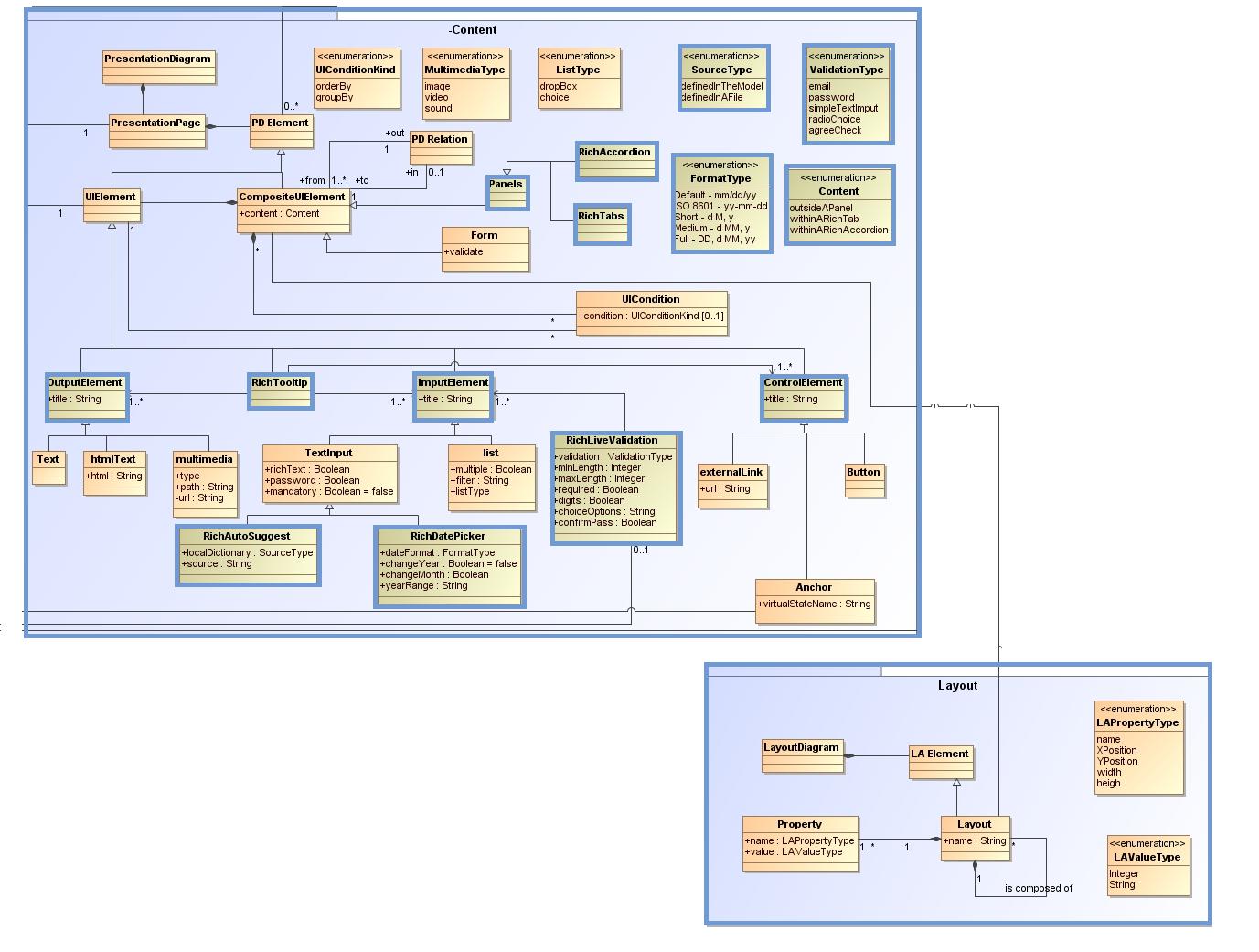


Figura 2 Metamodelo de contenido y estructura

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

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

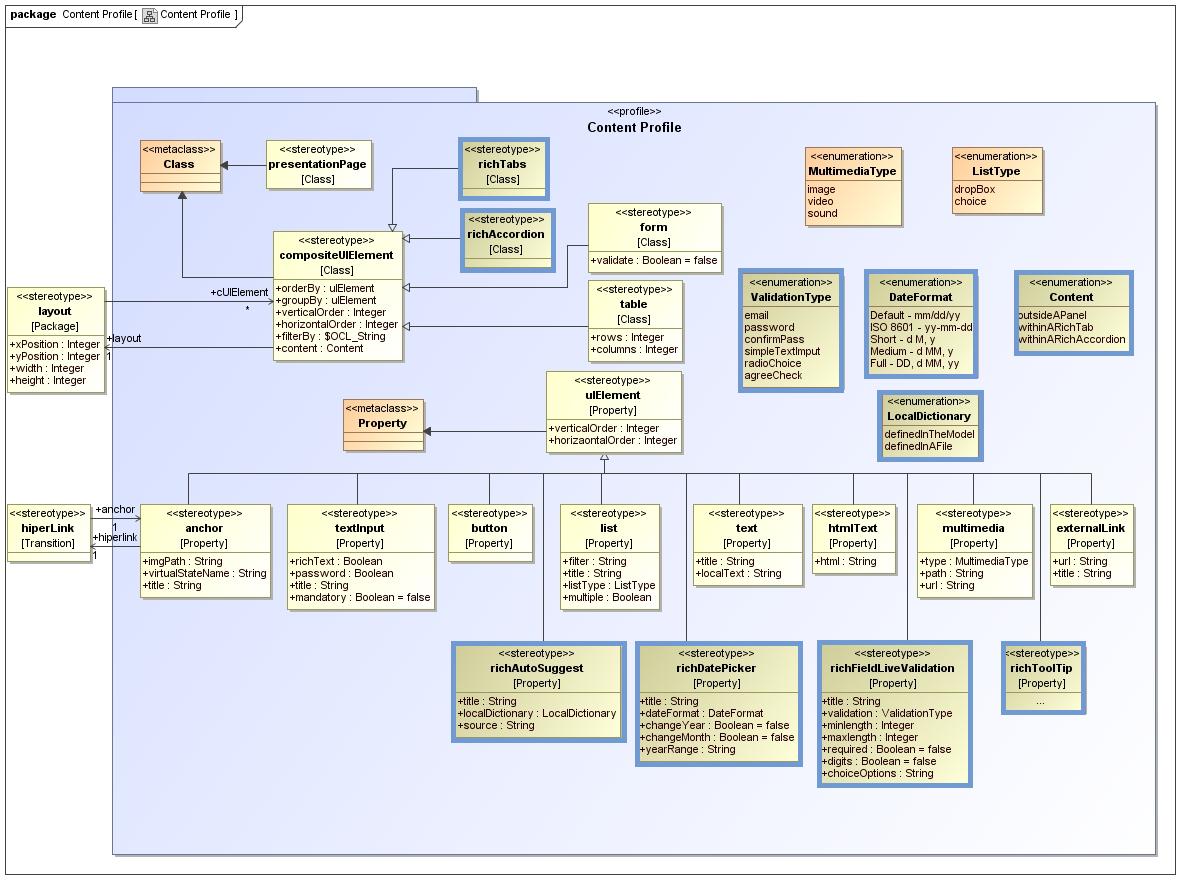
****

Figura 3 Perfil de contenido de MoWebA

**3.3.1 RichAutoSuggest**

Este elemento de interfaz enriquecido, contiene los valores etiquetados *localDictionary* y *source*. El primero corresponde a una enumeración que permite definir al diccionario de sugerencias en un archivo en formato .*xml* o bien definir el listado de sugerencias en el mismo diagrama de clases en un formato de cadenas de sugerencias separado por comas. El valor etiquetado *source* será completado de acuerdo a lo definido en la enumeración anterior. En el caso de estar definido con el valor *definedInTheModel*, entonces *source* que es del tipo *string*, contendrá un conjunto de palabras correspondiente a las sugerencias separadas entre comas. Si en la enumeración está definido el valor *definedInAFile*, entonces el diccionario de palabras está definido en un archivo xml y en valor etiquetado *source* se colocará la ruta en la cual se encuentra el archivo de palabras. El *richAutoSuggest*, como así también todos los elementos no compuestos de interfaz, poseen el valor etiquetado *title*, que representa al mensaje personalizado correspondiente a un *tooltip* nativo, común en los elementos HTML. Si el estereotipo *richToolTip* está definido en alguno de los atributos de un *compositeUIElement*, esto indicará que ese atributo será resaltado con un mensaje que está parametrizado en el valor etiquetado title del atributo, pero a diferencia de desplegar el tooltip nativo, se mostrará en pantalla un tooltip personalizado, al estilo jQueryUI.

**3.3.2 RichDatePicker**

Este elemento de interfaz enriquecido contiene a los valores etiquetados *dateFormat*, *changeYear*, *changeMonth* y *yearRange*. El *dateFormat* corresponde a una enumeración que contiene cinco formatos de fecha distintos que son *Default - mm/dd/yy*; *ISO 8601 - yy-mm-dd*; *Short - d M, y*; *Medium - d MM, y* y el *Full - DD, d MM, yy*. El valor etiquetado *changeYear* es un valor booleano que indica la ausencia o presencia de un rango de años desplegable en una lista que formará parte del *datePicker.* Por omisión, si *changeYear* está configurado en verdadero, se mostrará en el *datePIcker* una lista desplegable presentando los diez años anteriores a partir de la fecha actual. También es posible asignar al valor etiquetado *yearRange* un rango de años para el *richDatePicker* que se define en el formato yyyy:yyyy; por ejemplo 1970:2015. Definir *yearRange* resulta ideal para la selección de fechas pasadas, como el año de nacimiento o fechas históricas.

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

**3.3.3 RichToolTip**

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

**3.3.4 RichFieldLiveValidation**

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

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

Por otro lado, la validación del tipo *radioChoice*, permite establecer de un listado de opciones de selección, un valor mandatorio, que es establecido en el valor etiquetado *choiceOptions*, como cadenas separadas por comas. De todas las opciones existentes en *choiceOptions*, una de ellas se establece con la palabra *Required*, separada con un espacio en blanco para indicar que esa opción es mandatoria y debe seleccionarse de manera obligatoria.

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

**3.3.5 Form**

Este elemento de interfaz es una especialización de un *compositeUIElement* y ha sido extendido con el valor etiquetado booleano *validate*,que indica si el formulario en cuestión tendrá o no algún tipo de validación de campo. Por lo tanto *validate* debe estar establecido en verdadero para que el RichFieldValidation pueda entrar en vigencia.

**3.3.6 RichAccordion**

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

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

**3.3.7 RichTabs**

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

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

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

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

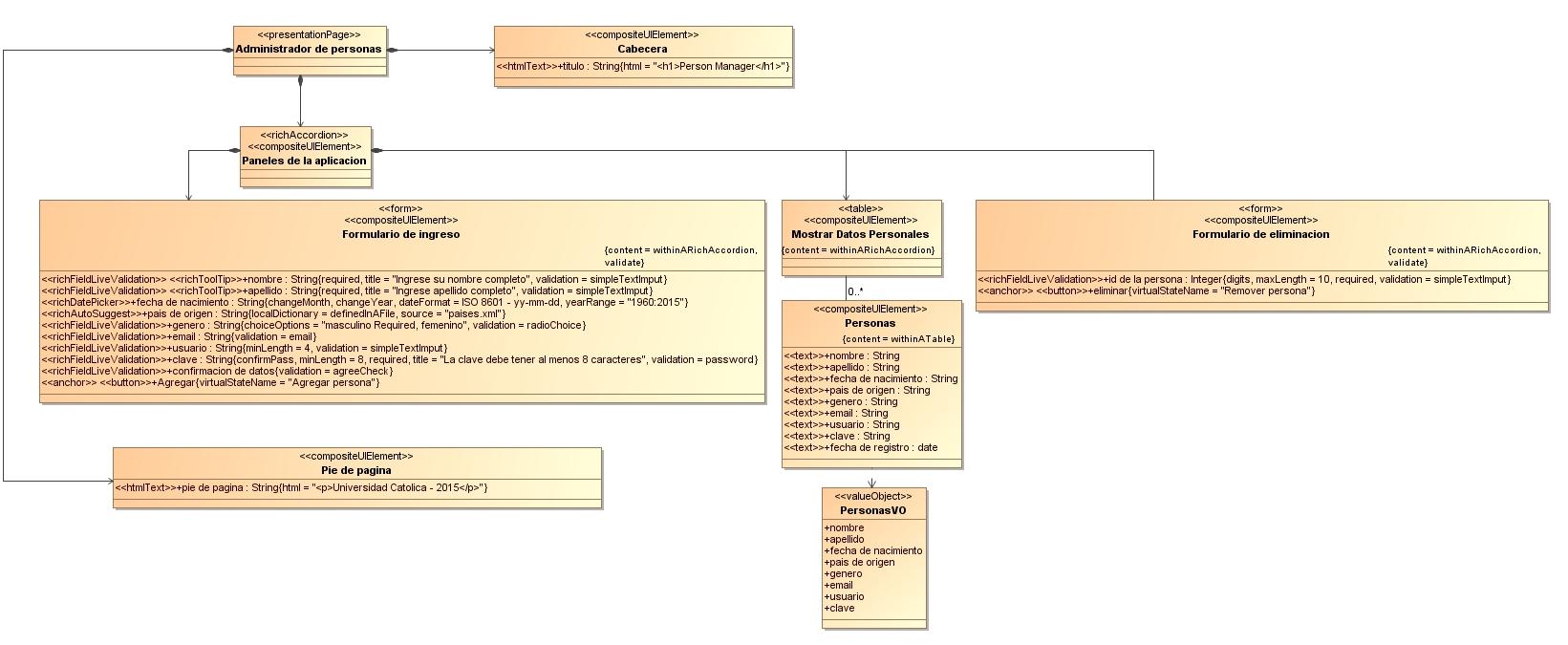


Figura 4 PIM de una aplicación de ejemplo con MoWebA

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

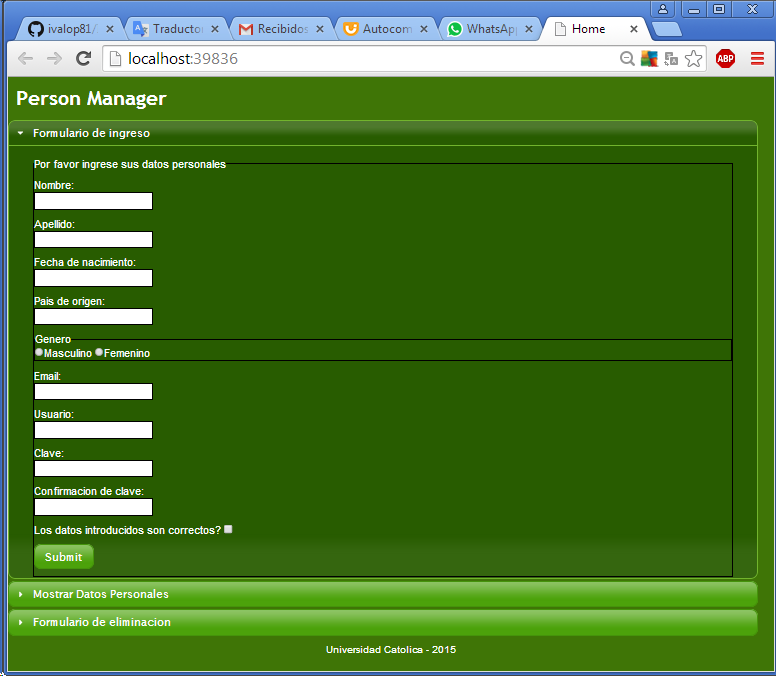


Figura 5 Interfaz final obtenida a partir del PIM de ejemplo



Figura 6 Elementos enriquecidos y validaciones de campos que forman parte de la aplicación

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

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

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

1. **No Magic:** <http://www.nomagic.com/products/magicdraw.html> 2015 [↑](#footnote-ref-1)
2. **Eclipse Modelling Framwwork:** <https://www.eclipse.org/modeling/emf> 2015 [↑](#footnote-ref-2)
3. **Acceleo:** <https://eclipse.org/acceleo> 2015 [↑](#footnote-ref-3)