**Capítulo 4**

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

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

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

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

En su definición original, la capa de presentación de *MoWebA* contiene diversos elementos para la interfaz de usuario que son de uso común en 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* y el *tooltip*) serán presentados en *MoWebA* como *richAccordion*, *richTabs*, *richAutoSuggest,* *richDatePicker*y *richToolTip.* El *live validation*, describe diversas validaciones a los campos de entrada y se presentan en *MoWebA* como *richMinLength*, *richMaxLength*, *richOnlyDigits*, *richConfirmPass* y *richEmail* respectivamente.

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

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

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



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

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

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

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

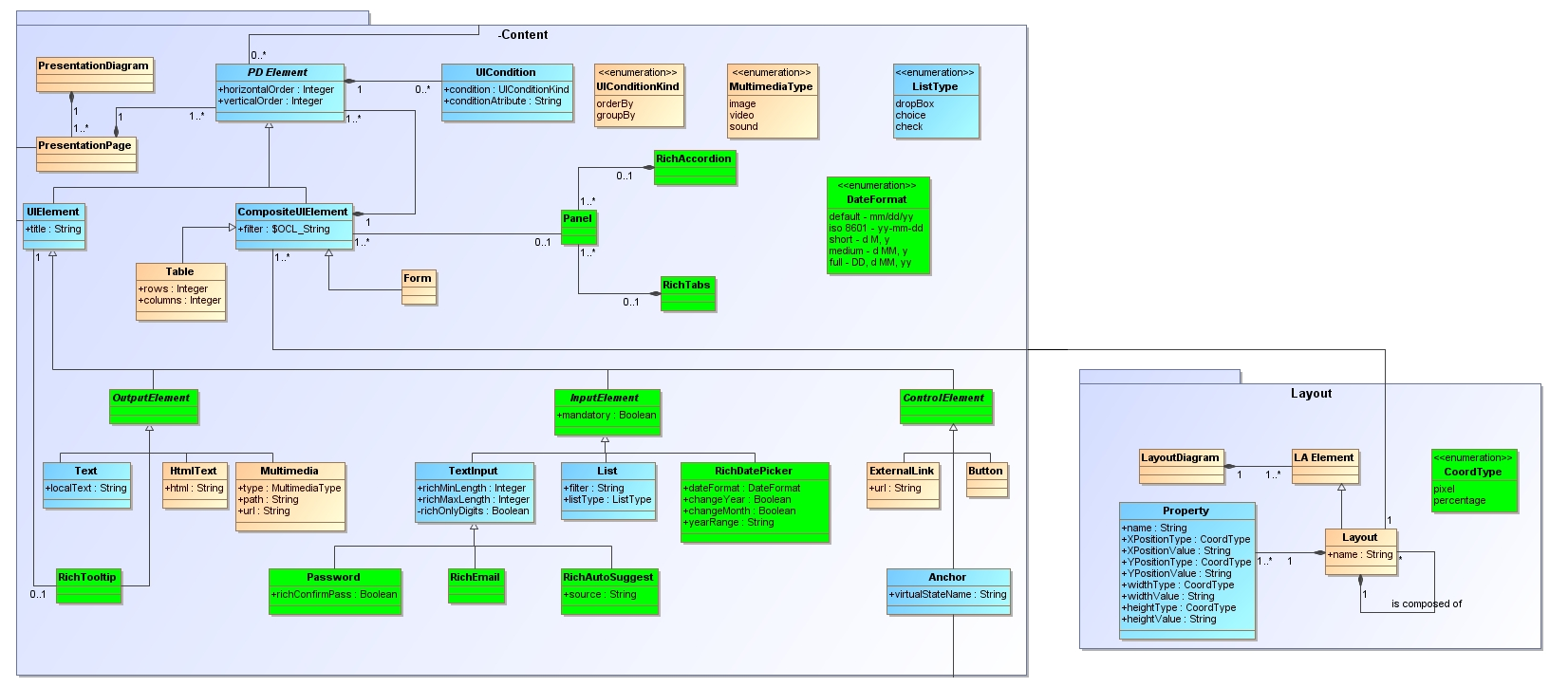
**4.2.1 El metamodelo de Contenido extendido**

Primeramente en el metamodelo de *Contenido*, se estableció una jerarquía entre los elementos compuestos (*CompositeUIElement*) y los elementos simples o elementos hoja (UIElements), aplicando el patrón *composite,* que es de uso común en el mundo de la ingeniería de software, principalmente cuando se desea desarrollar soluciones generales. El patrón *Composite* permite crear una jerarquía de elementos anidados unos dentro de otros. Cada elemento permite alojar una colección de elementos del mismo tipo, hasta llegar a los elementos “reales” que se corresponden con los nodos “Hoja” del árbol [<e2014>]. 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*.

****

**Figura 2** Metamodelo de Contenido y Estructura

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

**4.2.1.1 RichAutoSuggest**

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

<?xml version=**”1.0”** encoding=**”UTF-8”** standalone=**”no”**?>

<tags style=”MEDIUM”>

<tag>

<name>Paraguay</name>

</tag>

<tag>

<name>Portugal</name>

</tag>

<tag>

<name>Paquistan</name>

</tag>

.

.

.

<tag>

<name>. . .</name>

</tag>

</tags>

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



Figura 3 El *widget* *RichAutoSuggest*

**4.2.1.2 RichDatePicker**

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

\* ***Default - mm/dd/yy****:* Este formato es el valor por omisión de numerosas librerías *Javascript*. Por ejemplo, 06/08/2015

\* ***ISO 8601 - yy-mm-dd****:* Este formato es el ISO 8601 para el establecimiento de fechas. Por ejemplo, 2015-06-08

\* ***Short - d M, y****:* Este es un formato de fecha corta. Por ejemplo, 8 Jun, 15

\* ***Medium - d MM****:* Este es un formato de fecha mediana.Por ejemplo, 8 June, 15

\* ***Full - DD, d MM, yy****:* Este es un formato de definición de fecha completa.Por ejemplo, Monday, 8 June, 2015

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

Por último, el valor etiquetado booleano *changeMonth*  permite desplegar una lista con todos los meses del año para una rápida selección. En la se presenta el *widget* *RichDatePicker* con el formato de fecha *Default - mm/dd/yy****.***



Figura 4 El *widget RichDatePicker*

**4.2.1.3 RichToolTip**

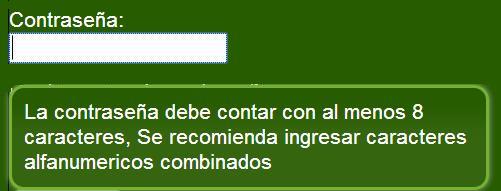
Este elemento de salida, tiene como objetivo enriquecer con mensajes personalizados a cualquiera de los elementos que forman parte de la clasificación de elementos de entrada, salida y control.

Figura 5 El *widget RichToolTip*

Al definirse este elemento en conjunción con algunos de los elementos simples de entrada, salida o de control, implica que un mensaje emergente será desplegado cuando el puntero del mouse se posicione sobre el elemento. Cada uno de los elementos de entrada, salida y control posee el valor etiquetado *title*, que corresponde al mensaje que será desplegado. En la se presenta el widget *RichToolTip*, desplegando un mensaje al posicionar el puntero del mouse sobre el campo Contraseña del tipo *Password*.

**4.2.1.4 Live Validation**

El *Live Validation* es un conjunto de extensiones que permite llevar a cabo validaciones locales a diversos elementos pertenecientes a un formulario. Estas validaciones pueden llevarse a cabo a diversos elementos de entrada, como a los del tipo *TextInput,* 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 se presentan algunos ejemplos de *Live Validation*.

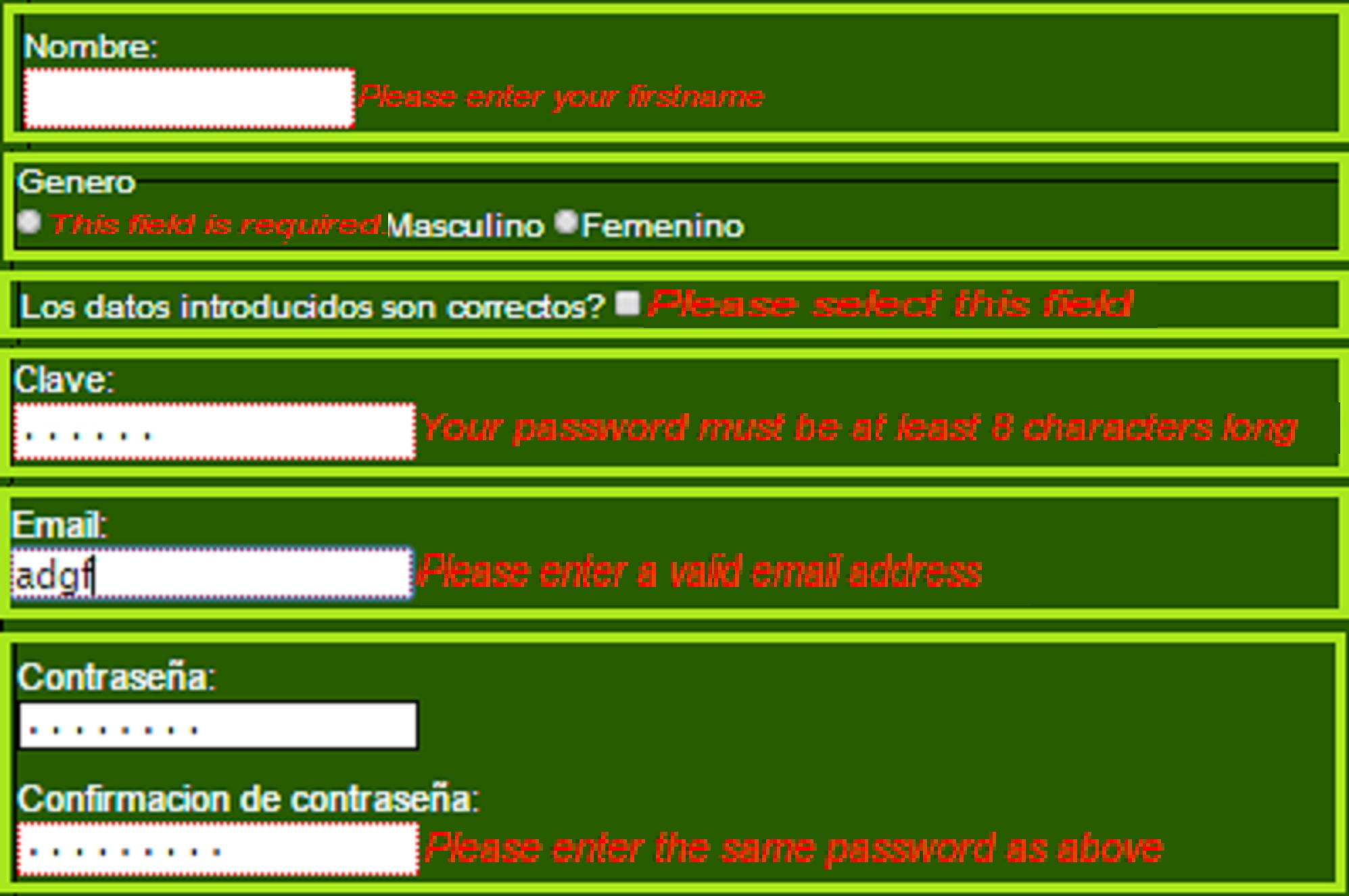


Figura 6 Ejemplos de *Live Validation*

**4.2.1.5 RichAccordion**

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

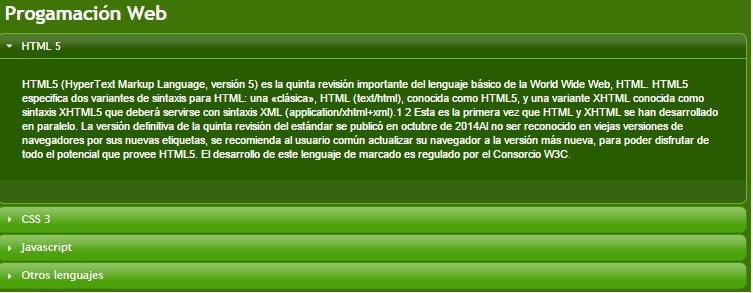


Figura 7 Ejemplo del *widget RichAccordion*

**4.2.1.6 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.



Figura 8 Ejemplo del *widget RichTabs*

**4.2.2 El metamodelo de Estructura extendido**

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

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

**4.3 - El perfil de contenido y estructura extendido.**

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

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

Con respecto al perfil de *Estructura*, el estereotipo *layout* que extiende a la metaclase Package, contempla todos los nuevos atributos agregados al metamodelo de *Estructura* presentados en la sección anterior. En la Figura 9 se presenta el perfil de Estructura extendido.

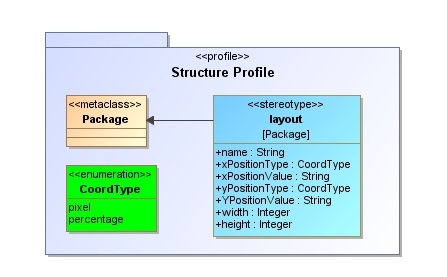


Figura 9 Perfil de Estructura extendido.

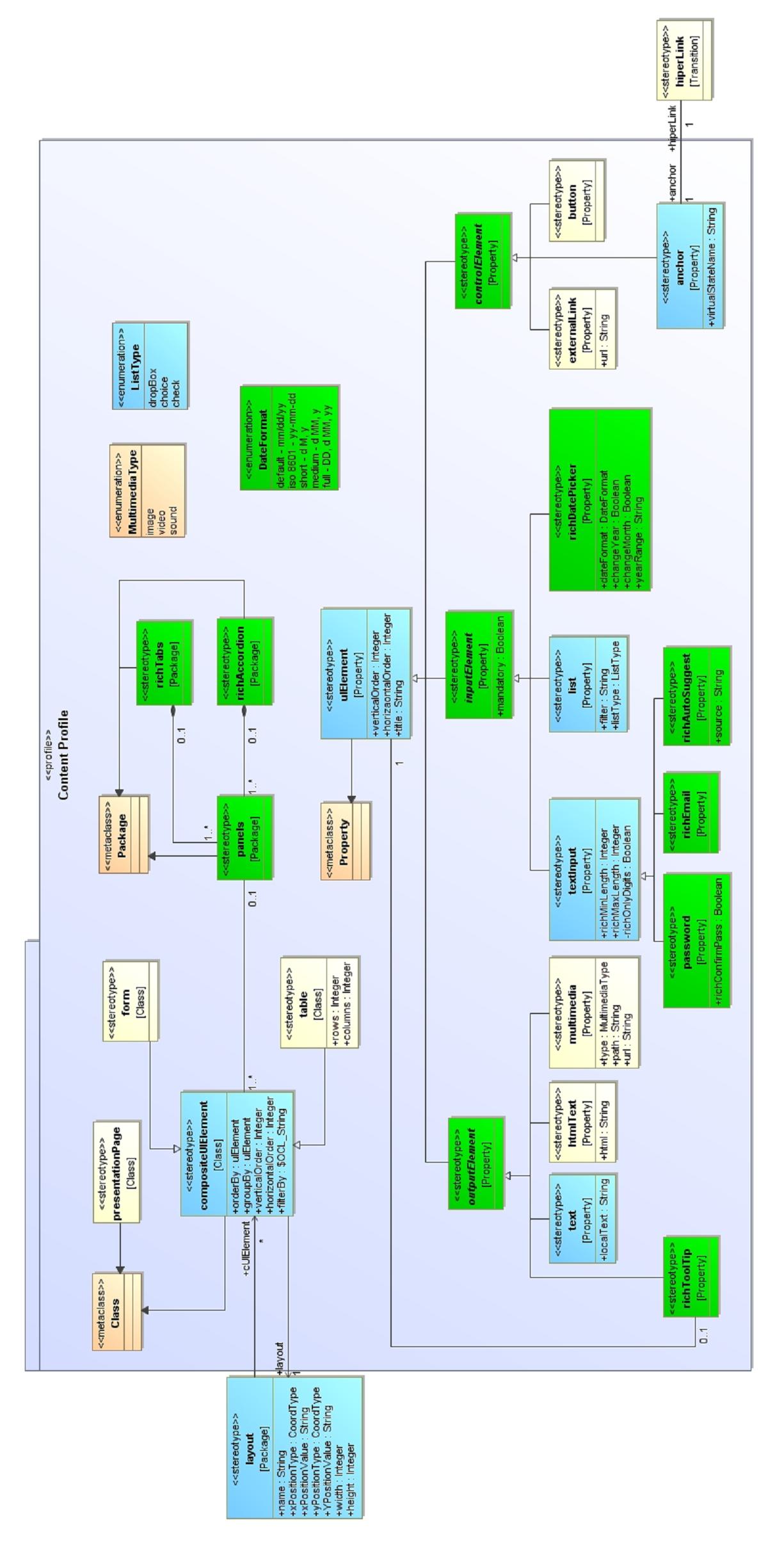
****

Figura 10 Perfil de contenido de *MoWebA*

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

En esta sección se presenta un ejemplo de modelado con las extensiones llevadas a cabo a *MoWebA*, con la idea de dar a conocer la manera en que se implementan los modelos independientes de la plataforma (*PIM*) de la propuesta de este trabajo de fin de carrera. En la se presenta el modelado del sistema *Person Manager* que se describe en el Anexo 1. Cada uno de los elementos que forman parte de la extensión a *MoWebA* se encuentran presentes en el *PIM* de Contenido se describen a continuación:

a) El *richAccordion que* está compuesto de tres paneles que son: Agregar persona, Listar personas y Eliminar personas, representadas con el estereotipo *Panels*. Cada uno de los *Panels* contiene un elemento compuesto *compositeUIElement* que engloba a varios elementos simples *uIElements.* En los paneles Agregar persona y Eliminar persona se encuentran varios elementos simples que forman parte de la extensión *RIA* a *MoWebA*.

b) Los campos *nombre* y *apellido* son campos con los estereotipos *textImput* y *richToolTip*. Ambos campos son obligatorios (*mandatory*) y despliegan mensajes que sugieren al usuario que se ingrese el nombre y el apellido completo.

c) El campo fecha de nacimiento tiene el estereotipo *richDatePicker* que tiene los valores etiquetados *changeMonth* y *changerYear* activados. El formato de fecha elegido es el *default (mm-dd-yyyy)*  y el rango de años desplegable (yearRange) se encuentra en el rango de 1970 al 2015.

d) El campo país de origen está representado con el estereotipo *richAutoSuggest* y contiene la lista de sugerencias, en el archivo países.xml configurada en el valor etiquetado *source*.

e) El campo género (con las opciones masculino, femenino definido en el valor etiquetado *filter*) del tipo *list*, es un campo de selección (*choice)* que debe seleccionarse obligatoriamente (*mandatory*).

f) El campo email con el estereotipo *richEmail*, valida que el formato del campo sea del tipo email.

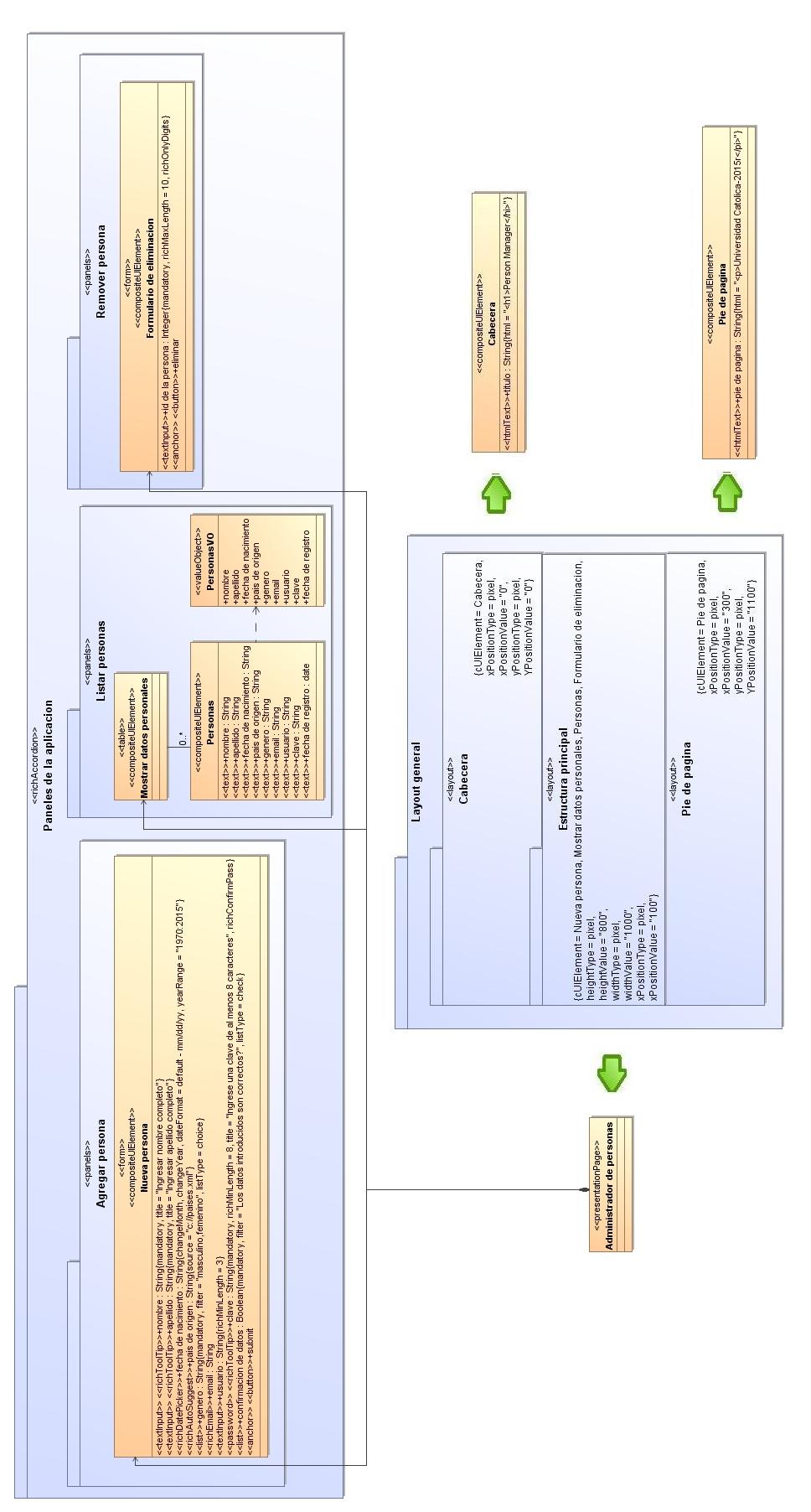
g) El *texImput* usuario tiene en cuenta que la longitud del campo sea de al menos tres caracteres (*minLength*=3).

h) El campo clave tiene establecido los estereotipos *password* y *richToolTip*. La clave es de ingreso obligatorio (*mandatory* = *true*) y debe reconfirmarse (*confirmPass*=*true*). El mensaje “Ingrese clave de al menos 8 caracteres” establecido en *title* será desplegado al posicionar el *mouse* sobre el campo clave.

i) El campo confirmación de datos con el estereotipo *list* es un (check) que debe marcarse obligatoriamente para confirmar los datos ingresados.

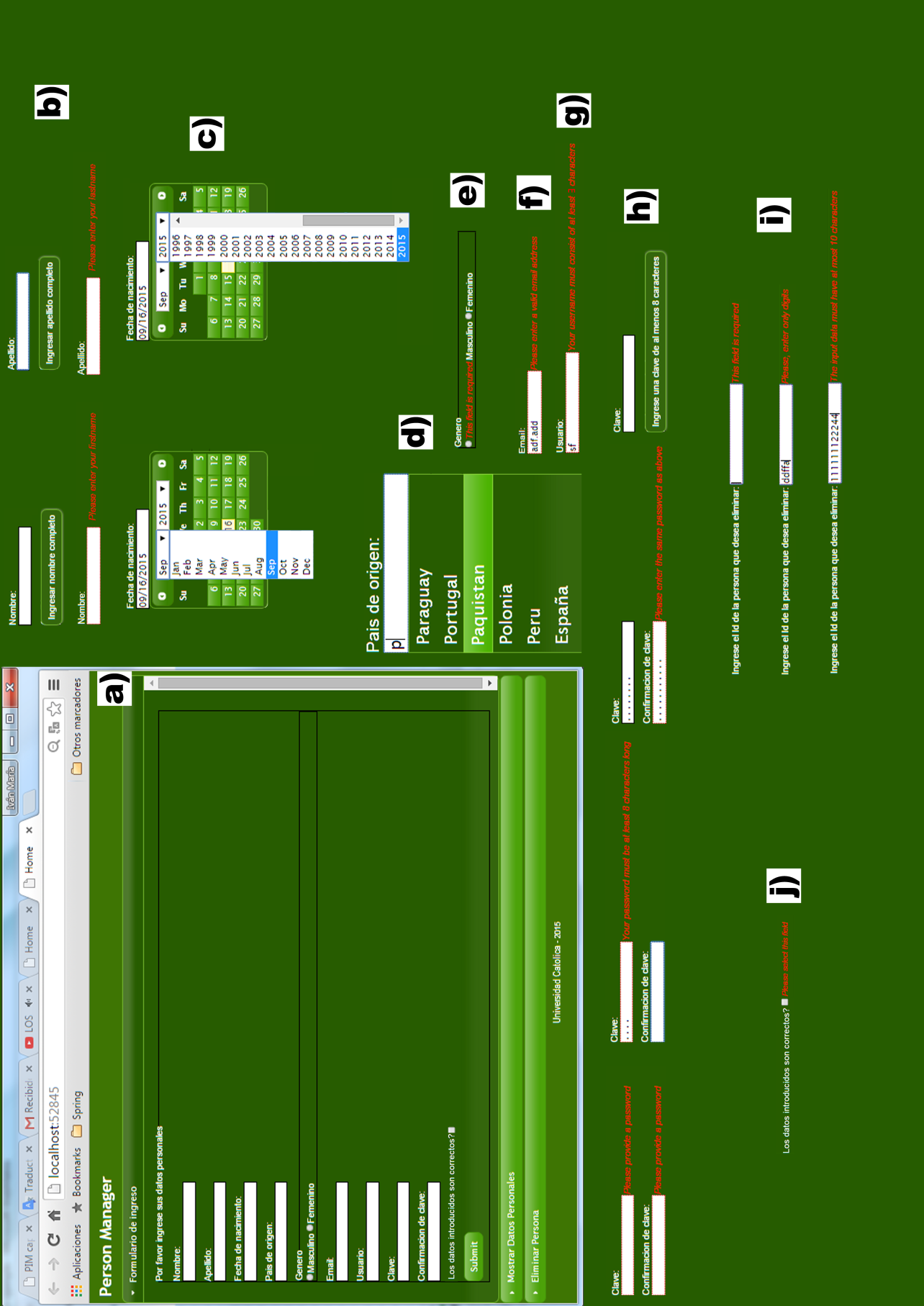
j) En el Formulario de eliminación, del panel Eliminar persona, se encuentra el *textImput* campo id de la persona, que es de ingreso obligatorio, permite solamente dígitos enteros positivos (*richOnlyDigits*) con un tamaño máximo de diez (*maxLength* = 10).

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



**Figura 11** *PIM* del Person Manager con *MoWebA*

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



**Figura 12** Interfaz final del *Person Manager* obtenida a partir del *PIM*.

**4.5 La Transformación de modelo a texto (*M2T*) para los diagramas de contenido y estructura de *MoWebA***

En esta sección se presentan algunos conceptos básicos acerca de la generación de código y los beneficios que aportan al proceso de desarrollo basado en modelos. El proceso *M2T* puede tener varios objetivos finales, como la generación, a partir de los modelos de documentación, código fuente, o cualquier componente de software necesario en una aplicación. Aquí nos concentraremos en la generación de código fuente para la interfaz de usuario de una aplicación *Web* representada por medio de modelos que han sido definidos en *MoWebA*. La herramienta *M2T* *Acceleo* es la herramienta utilizada para la generación de código.

Seguidamente se presentará a las plantillas (*templates*) para llevar a cabo la transformación de los *PIM* de entrada de *MoWebA* correspondiente al contenido y a las plantillas de transformación para establecer la configuración de las posiciones de cada uno de los elementos del *PIM*. En el Anexo 2 del libro se encuentra la plantilla completa de Contenido. En la siguiente sección de este capítulo se presenta la plantilla de Estructura.

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

**4.5  LA HERRAMIENTA DE TRANSFORMACIÓN *M2T* *ACCELEO***

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

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

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

En el enfoque tomado en este trabajo, la generación de código es total a partir de los modelos de los *PIM* de entrada, para los elementos de la capa de presentación de *MoWebA*. El código fuente a generarse a partir de los modelos será *HTML*, *Javascript* para la representación de los distintos elementos de interfaz de *MoWebA* como así también *CSS* para el posicionamiento de estos elementos en las páginas.

**4.6 Transformacion a codigo de los *PIM* de *MoWebA* con *Acceleo*.**

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

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

* *IDE* Eclipse Kepler Service release 2
* *Acceleo* Versión 3.4
* *UML* Designer for Eclipse Kepler version 3.0

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

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

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

La plantilla de *Estructura* transforma las posiciones definidas en pixeles o en porcentajes, a cada uno de los *compositeUIElement* en un archivo .*CSS* con las coordenadas de posicionamiento correspondiente a cada uno de ellos. A continuación se presentaran las plantillas de Contenido y Estructura respectivamente.

**4.6.2 Plantilla de transformación para los elementos del perfil de Contenido.**

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

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

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

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

El estereotipo *richDatePicker* indica que el atributo *fecha de nacimiento*, es un calendario y los valores etiquetados del atributo, definen las características del *datePicker*. Dentro de los valores etiquetados definidos para el atributo *fecha de nacimiento*, tenemos a *changeMonth*, *dateFormat* y *yearRange;* que indican respectivamente que una lista de los meses se agregará al *datePicker*, que el formato de fecha con el cual el cuadro de texto será completado, luego de la selección de una fecha dada en el calendario desplegado, será del tipo *default –mm-dd—yy* y que el rango 1970:2015 será desplegado en una lista.



Figura 13 *PIM* de contenido de *MoWebA* y *templates* de trasnformación de Contenido para el *richDatePicker*

La plantilla de contenido lleva a cabo dos iteraciones completas sobre las clases y sus atributos respectivos definidos en el *PIM* de presentación. La primera pasada, es para la generación del código *Javascript* correspondiente a los elementos enriquecidos. La otra pasada es para la definición del cuerpo del elemento que corresponde al *tag body*. En cada una de las iteraciones sobre las clases, se verifican sus atributos estereotipados para que en caso de estar definido en el modelo, un elemento en particular, se escriba el código correspondiente al elemento. El nombre del atributo estereotipado de un elemento enriquecido de la clase, se interpreta en la plantilla como el identificador (id) de elemento, y sirve para machear el comportamiento dinámico del elemento con la definición del mismo. Cada uno de los metamarcadores, iteradores y sentencias condicionales, permiten obtener los valores del modelo de entrada, iterar sobre los distintos elementos y preguntar sobre los distintos elementos de interfaz. En la se observa el código *HTML* generado por la plantilla de transformación para el elemento *richdatePicker* definido en el ejemplo de la **Figura** 13.

****

Figura 14 Código fuente *HTML* generado para el *richDatePicker* generado a partir de las plantillas

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

Con la plantilla de transformación para el posicionamiento de los elementos de interfaz dentro de las páginas, es posible generar el código correspondiente a los *cascading style sheets* (*CSS*) a partir de los modelos *PIM* de estructura (*Layout*). Primeramente al igual que en la plantilla de contenido presentada anteriormente, es necesario importar los servicios Java para poder utilizar dentro de la plantilla, expresiones que no son *OCL* estándar, como por ejemplo el método *hasStereotype* que permite saber si un elemento *UML* posee cierto estereotipo para llevar a cabo decisiones. Seguidamente se decide el nombre y la extensión del archivo de salida por medio del *tag file* y dentro de este *tag* comienza el proceso de recorrido dentro los elementos del tipo *package*, en donde se buscan los valores etiquetados del tipo *cUIElement*. Para cada uno de los valores etiquetados *cUIElement* encontrados dentro de unpaquete estereotipado con *Layout*, se agregan los valores correspondientes a las posiciones definidos en el modelo *PIM*. Las posiciones a definirse corresponden a los valores en pixeles del *height*, *width*, *xPosition(left)* y *yPosition(top)*. En la Figura 15 se presenta el *template* de transformación de Estructura.



Figura 15 Plantilla de transformación para el posicionamiento de elementos

**4.7 Resumen del capitulo**

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

Todos estos nuevos elementos primeramente fueron agregados al metamodelo de *Contenido* para la representación de la sintaxis abstracta de cada uno de ellos. A partir de esta definición, se presentó el perfil de Contenido, que extiende a *UML* permitiendo expresar la sintaxis concreta de *MoWebA*. En el perfil de Contenido, se describió cada uno de los nuevos elementos agregados, con el detalle de cada uno de sus valores etiquetados, que son necesarios para expresar las características que van a tener los *widgets*, como así también el elemento de validación de campos. También se presentó el perfil de *Estructura*, en donde se mostró como se establecen las coordenadas de cada uno de los elementos. Seguidamente se ilustró un ejemplo de *PIM* con las extensiones *RIA* propuestas a *MoWebA*, junto a algunas vistas de tomas de pantalla de la aplicación.

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