MDD+

Mejorando el Proceso de Desarrollo de Software: Propuesta basada en MDD

Nro. Referencia: 14-INV-056

Investigador Principal: Luca Cernuzzi, Universidad Católica “Nuestra Señora de la Asunción”

Un Enfoque MDD para el Desarrollo de RIA

Informe sobre la Etapa de Diseño

Autor del Documento: Guido Nuñez, Universidad Católica “Nuestra Señora de la Asunción”

Noviembre – 2016

Este proyecto es financiado por el CONACYT a través del programa PROCIENCIA con recursos del Fondo para la Excelencia de la Educación e Investigación (FEEI) del FONACIDE.

Tabla de contenido

1. Introducción 2
2. RIA a generar 2
3. Proceso de desarrollo de RIA 3
4. Extensiones al PIM 3
   4.1. Extensiones al diagrama de entidades 4
   4.2. Extensiones al diagrama lógico 5
   4.3. Extensiones al diagrama de contenido 6
5. ASM RIA 7
   5.1. Almacenamiento de datos en el cliente 7
   5.2. Lógica de negocios en el cliente 8
   5.3. Comunicación asíncrona entre cliente y servidor 9
6. Modelo de una RIA 10
   6.1. CIM/PIM 10
   6.2. ASM/PSM 13
7. Referencias 16

1. Introducción

Este trabajo ha sido desarrollado con el soporte de CONACYT, en el contexto del proyecto "Mejorando el proceso de desarrollo de software: propuesta basada en MDD", que busca explorar los beneficios de la aplicación del Desarrollo Dirigido por Modelos (MDD - Model-Driven Development) [1] en el desarrollo de aplicaciones software de buena calidad, utilizando tecnologías actuales.

El presente trabajo busca proveer un enfoque MDD que facilite el proceso de desarrollo de una Aplicación Enriquecida de Internet (RIA – Rich Internet Application) [2]. Para ello se parte de MoWebA [3], un enfoque MDD concebido para el desarrollo de aplicaciones web, el cual, es extendido mediante metamodelos y perfiles de modo a proporcionar funcionalidades RIA. La RIA a generar tiene en cuenta las características de almacenamiento de datos en el cliente, lógica de negocios en el cliente y comunicación asíncrona entre cliente y servidor [4][5]. Se han introducido elementos de modelado que representan funcionalidades RIA que abarcan estas características. Se trabajó sobre los diagramas de entidades, lógico y de contenido de MoWebA, modificando o agregando elementos correspondientes al PIM que faciliten el posterior modelado de elementos RIA en el ASM. Seguidamente, se elaboraron estos elementos RIA abarcando las características recientemente mencionadas.

En este documento se presentará la RIA a generar, detallando las funcionalidades a ser implementadas. Seguidamente se presentará el proceso a seguir para desarrollar la RIA. Luego se introducen metamodelos y perfiles que representan extensiones realizadas al PIM y el ASM. Finalmente se presenta a través de un ejemplo ilustrativo un modelo de una aplicación que hace uso de las extensiones propuestas.

2. RIA a generar

El enfoque busca generar una RIA que contemple puntualmente las características de almacenamiento de datos en el cliente, lógica de negocios en el cliente y comunicación asíncrona entre cliente y servidor.

Se tienen en cuenta estas características debido a que ellas son de gran importancia para el despliegue de la aplicación ya que representan las operaciones llevadas por debajo para la correcta presentación de la interfaz de usuario.

Considerando el almacenamiento de datos en el cliente, se introducen los elementos *ClientValueObject* y *ClientStaticObject* que representan variables almacenadas del lado del cliente. *ClientValueObject* obtiene datos de una base de datos en el servidor, mientras que *ClientStaticObject* obtiene datos especificados estáticamente en el modelo de la aplicación. Ambos permiten especificar el nivel de persistencia de los datos, que pueden ser del tipo temporal o permanente.

Para la lógica de negocios en el cliente, se añaden los elementos *RichTable*, *RichForm* y *RichTextInput* que permiten especificar elementos de interfaz de usuario con operaciones de lógica de negocios que serán ejecutadas en el cliente. *RichTable* consiste en una tabla con operaciones de paginación, ordenamiento y búsqueda ejecutadas en el cliente (aunque también puede ser del lado del servidor); *RichForm* es un formulario con opción de autocompletado de sus campos basados en valores previamente ingresados; y *RichTextInput* representa una entrada de texto que posee también la opción de autocompletado anterior, pero además permite especificar una lista de valores a sugerir.

En cuanto a la comunicación asíncrona entre cliente y servidor, se adhiere el elemento *AsynchronousCall* que consiste en una invocación asíncrona de un servicio desde un elemento de presentación, permitiendo especificar el evento ocurrido sobre el elemento, el tipo de solicitud, tipo de respuesta y parámetros a enviar al servicio.

3. Proceso de desarrollo de RIA

Con el fin de modelar y posteriormente generar una RIA se parte haciendo uso de la herramienta MagicDraw[1]. Mediante la importación en la herramienta de los perfiles de MoWebA y del perfil RIA propuesto es posible realizar el modelo de una RIA que aproveche las diversas funcionalidades proporcionadas por este trabajo. Este modelo es exportado a un archivo en formato XMI, el cual a su vez es importado en la herramienta Acceleo[2]. Esta herramienta se vale de plantillas con reglas de transformación propuestas que permiten realizar transformaciones modelo a texto generando el código final correspondiente a la implementación de la RIA.

La implementación contiene código HTML5 para elementos de interfaz de usuario y código Javascript para el comportamiento de dichos elementos. Además, las funcionalidades de HTML5 LocalStorage[3] y SessionStorage[4] son aprovechadas para el almacenamiento de datos del lado del cliente. JQuery[5] con sus plugins Datatables[6] y jQuery UI[7] son utilizados para la lógica de negocios del lado del cliente, generando tablas enriquecidas y autocompletado de entradas de texto. JQuery también es utilizado para implementar llamadas asíncronas entre cliente y servidor. El producto final consiste en una RIA que puede ser fácilmente alojada y desplegada en un servidor web. El proceso propuesto puede observarse en la Figura 1.

Figura 1: Proceso de modelado y generación de RIA.

4. Extensiones al PIM

A fin de facilitar el modelado de funcionalidades RIA, se han implementado extensiones al metamodelo y perfiles originales de MoWebA. Estas extensiones corresponden a funcionalidades que no se incluían en la versión inicial de MoWebA y que permiten facilitar el posterior modelado de características RIA. Las extensiones han sido aplicadas a los diagramas de entidades, lógico y contenido. Para facilitar la comprensión, las clases en los metamodelos y perfiles han sido coloreadas y distinguidas, usando el color salmón para aquellas que no han sufrido modificación en su forma original, el azul para las que han sido modificadas ya sea mediante la agregación, actualización o eliminación de alguna de sus propiedades y el verde para las nuevas clases adheridas.

[1] MagicDraw. http://www.nomagic.com/products/magicdraw.html
[2] Acceleo. https://eclipse.org/acceleo/
[3] LocalStorage. https://www.w3.org/TR/webstorage/#the-localstorage-attribute
[4] SessionStorage. https://www.w3.org/TR/webstorage/#the-sessionstorage-attribute
[5] JQuery. https://jquery.com/
[6] Datatables. https://datatables.net/
[7] JQuery UI. https://jqueryui.com/

4.1. Extensiones al diagrama de entidades

El diagrama de entidades original de MoWebA, utilizado para definir la estructura y relaciones estáticas entre clases del dominio del problema, fue extendido con el fin de proporcionar un mecanismo para el modelado del diagrama conceptual de una base de datos.

La extensión propuesta puede observarse en la Figura 2 y en la Figura 3. Ahora, una entidad (*Entity*) puede mapearse a una tabla de base de datos y estar compuesta de propiedades de entidad (*EntityProperty*) que permiten especificar: un tipo de dato (*dataType*), que puede tomar los valores *char*, *varchar*, *text*, *int*, *date*, *time*, *datetime*, un tamaño máximo (*size*) que puede ser aplicados a los tipos de datos *char*, *varchar* e *int*, y restricciones para evitar que una propiedad tome el valor nulo (*notNull*), para asegurar que cada registro de la tabla tenga un valor único (*unique*), para otorgar una clave primaria a cada registro (*primaryKey*), un valor por defecto (*defaultValue*) y condiciones específicas que el valor de una columna debe cumplir (*checkCondition*).

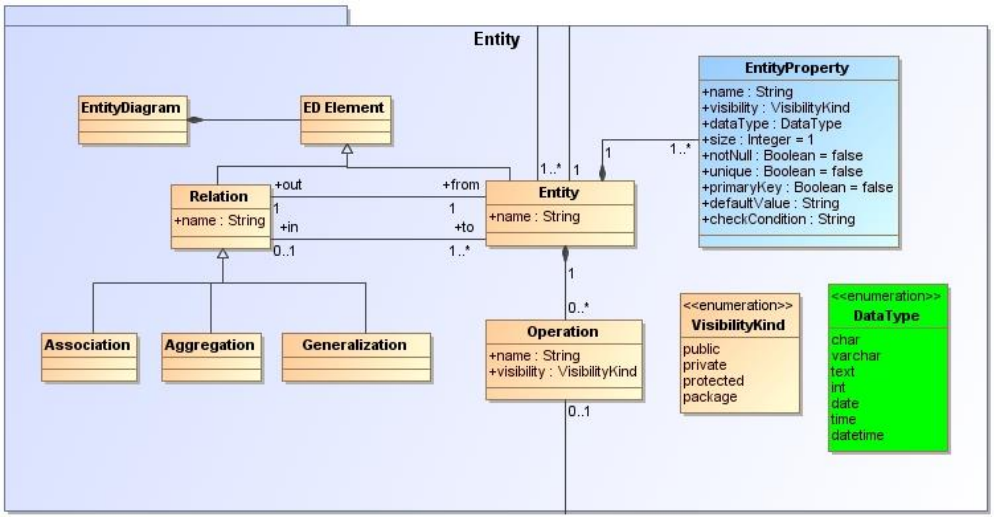

Figura 2: Metamodelo de entidades extendido.

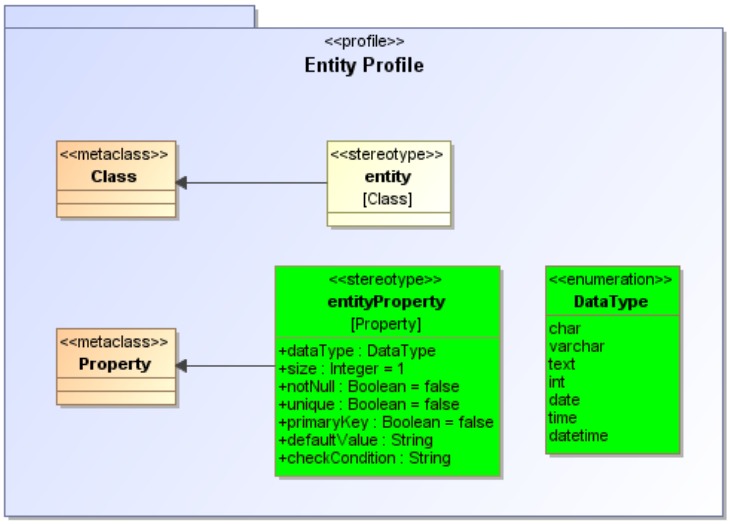

Figura 3: Perfil de entidades extendido.

4.2. Extensiones al diagrama lógico

El diagrama lógico de MoWebA permite la definición de objetos de valor (*ValueObjects*) que proveen visibilidad y acceso al dominio a la capa de presentación. Estos objetos de valor encapsulan datos, que dependen de una o más entidades, conteniendo un conjunto de atributos de las entidades dependientes.

La extensión propuesta de la Figura 4 y Figura 5 presenta una nueva clase que representa a un objeto estático (*StaticObject*). Este objeto es también utilizado para encapsular datos, pero a diferencia de los objetos de valor no dependen de alguna entidad. Un objeto estático es una clase compuesta por un conjunto de valores, los cuales son definidos de forma estática como propiedades de la clase. Estos valores poseen un nombre (*name*) utilizado como el valor en sí, un título (*title*) utilizado como texto a desplegar y un campo para pre-seleccionar el valor cuando es desplegado en una lista (*checked*).

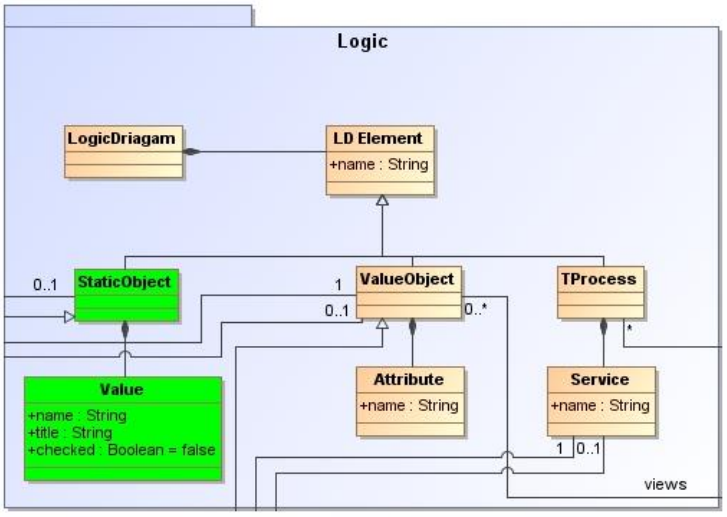

Figura 4: Metamodelo lógico extendido.

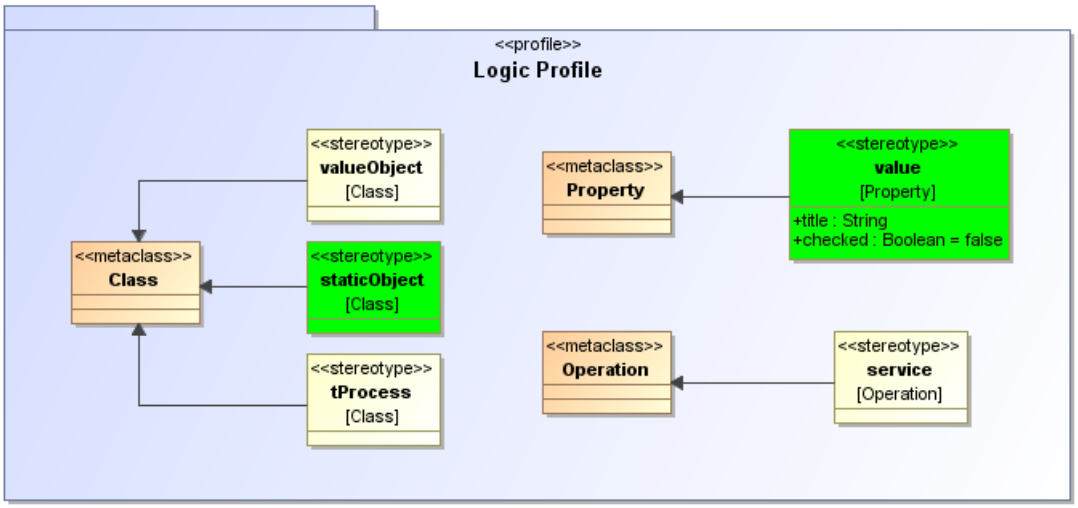

Figura 5: Perfil lógico extendido.

4.3. Extensiones al diagrama de contenido

El diagrama de contenido de MoWebA, utilizado para presentar a los usuarios los distintos elementos de cada página de la aplicación, también sufrió ligeras modificaciones, éstas pueden observarse en la Figura 6 y Figura 7.

Primeramente, la clase formulario (*Form*) ahora comprende un atributo que especifica la forma de envío de datos o tipo de solicitud (*requestType*) realizada al servicio encargado del procesamiento del formulario. Este atributo puede utilizarse para recuperar datos remotos (*retrieve*) o para insertar o actualizar datos remotos (*insert*/*update*). A nivel de implementación estos valores serán mapeados con los métodos GET y POST de formularios HTML.

Al elemento lista (*List*) se le añadió el atributo nombre (*name*), utilizado para agrupar sus elementos. Los elementos de lista del mismo grupo deben tener el mismo nombre.

El campo *default* de la enumeración formato de fecha (*DateFormat*) ha sido cambiado al formato *dd/mm/yyyy* que se corresponde al formato por defecto de las entradas de fechas en HTML.

Se ha eliminado la relación entre el elemento enlace o ancla (*Anchor*) con el hiperenlace (*Hyperlink*) del diagrama de nodos. En su lugar, el *Anchor* ahora se relaciona directamente con el estado virtual (*VirtualState*) o el estado de servicio (*ServiceState*) del diagrama de nodos al cual redirige el enlace. (este cambio no se encuentra coloreado debido a que afecta a las relaciones de la clase y no a sus atributos).

Por último, se agregó un botón de envío (*SubmitButton*) como especialización de la clase botón presente (*Button*), utilizado dentro de un formulario para enviar los datos de él a un servicio. Este botón se relaciona

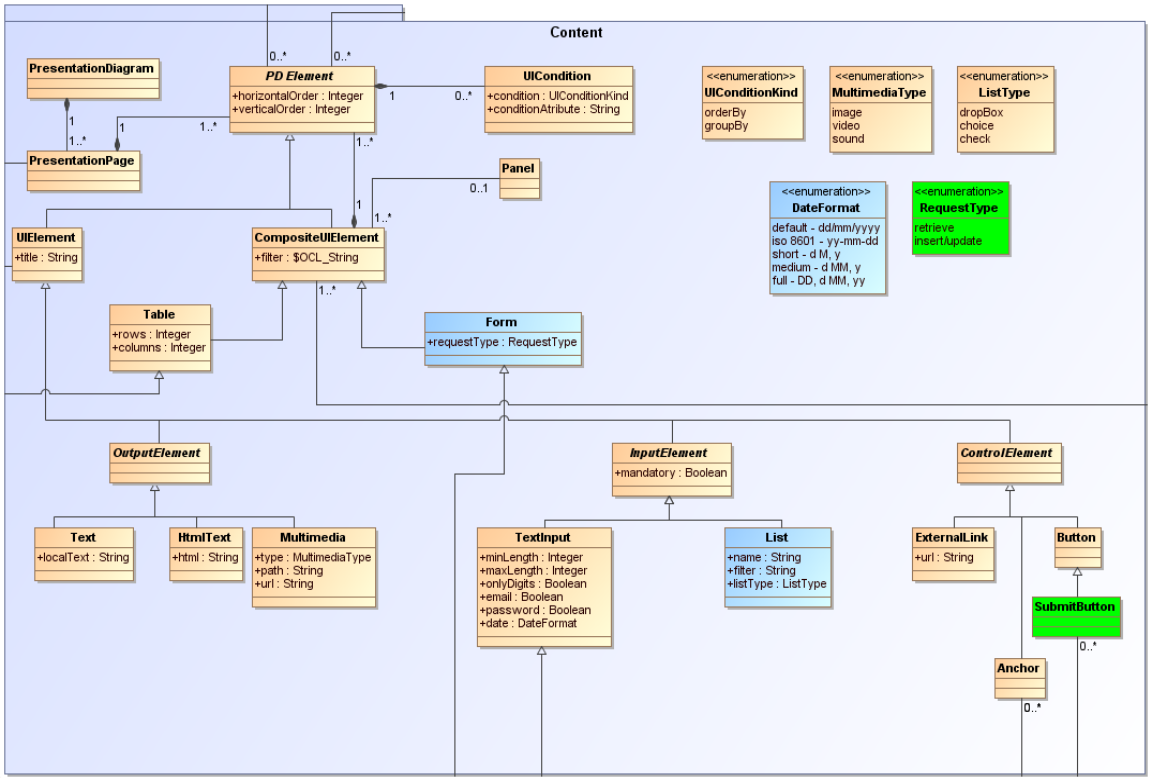

Figura 6: Metamodelo de contenido extendido.

con un servicio del diagrama de nodos para indicar el servicio encargado de procesar los datos enviados por el formulario.

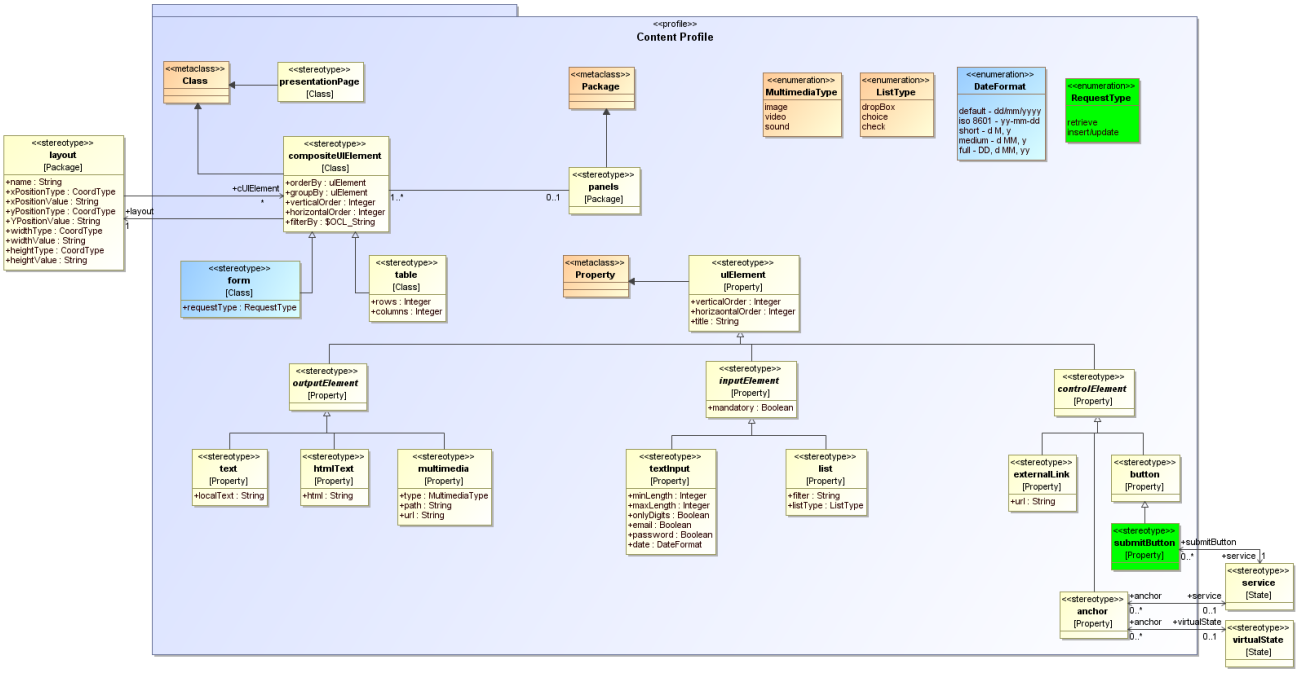

Figura 7: Perfil de contenido extendido.

5. ASM RIA

A partir de las extensiones propuestas en las secciones anteriores, ahora es posible presentar las extensiones que definen las funcionalidades RIA para MoWebA. Las funcionalidades abarcan las características de almacenamiento de datos en el cliente, lógica de negocios en el cliente y comunicación asíncrona entre cliente y servidor, y se presentan en el metamodelo RIA de la Figura 8 y el perfil RIA de la Figura 9. Los nuevos elementos especializan o se relacionan con elementos presentes en los diagramas lógico y de contenido.

A continuación se presentarán los nuevos elementos según la característica RIA que describen.

5.1. Almacenamiento de datos en el cliente

Para guardar datos en un cliente web, específicamente en un navegador web, se han creado dos nuevas clases, un objeto estático en el cliente (*ClientStaticObject*) y un objeto de valor en el cliente (*ClientValueObject*), ellos extienden al objeto estático y al objeto de valor del diagrama lógico respectivamente.

Los nuevos objetos se diferencian de los originales a nivel de implementación, ya que tanto el *ClientStaticObject* como el *ClientValueObject* serán mapeados a variables almacenadas en el navegador. El nombre de la clase corresponde al nombre de la variable y los valores de la clase con sus valores etiquetados asociados (*title* y *checked*) dan lugar a los valores de la variable. Además, estas nuevas clases permiten especificar un nivel de persistencia (*persistence*) que puede ser permanente (*permanent*) o temporal (*temporary*) permitiendo que la variable persista o no al término de una sesión o cierre del navegador. Si el nivel de persistencia es permanente se generará el objeto Localstorage de HTML, mientras que si la persistencia es temporal se generará el objeto Sessionstorage de HTML. Estos objetos serán generados tanto

para el *ClientStaticObject* como para el *ClientValueObject*, siendo la propiedad de persistencia la que decida cuál será el objeto a generar.

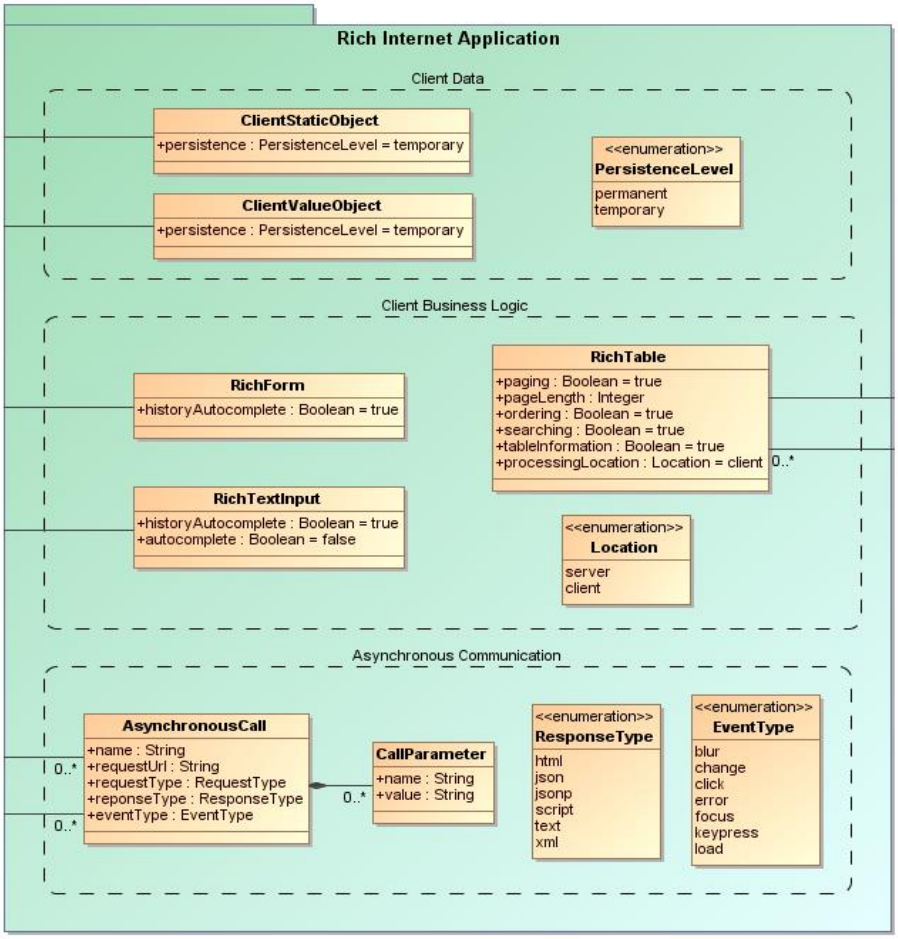

Figura 8: Metamodelo RIA.

5.2. Lógica de negocios en el cliente

En cuanto a procesos ejecutados en el cliente, se han creado tres elementos que se ejecutan en el navegador web, el formulario enriquecido (*RichForm*), la entrada de texto enriquecida (*RichTextInput*) y la tabla enriquecida (*RichTable*).

El formulario enriquecido consiste en una especialización del elemento formulario (Form) del diagrama de contenido. Contiene el atributo de autocompletado basado en historia (historyAutocomplete), que permite al navegador sugerir valores para completar las entradas del formulario, basados en valores previamente ingresados. A partir de cada instancia de un formulario enriquecido se generará un formulario HTML con la propiedad de autocompletado habilitada o no según lo establecido en el modelo.

La entrada de texto enriquecida extiende a la entrada de texto (*TextInput*) del diagrama de contenido. Así como el formulario enriquecido, la entrada de texto enriquecida posee el atributo de autocompletado basado en historia, pero además introduce el atributo de autocompletado (*autocomplete*) que se diferencia del anterior en que los valores a sugerir los obtiene a partir de una asociación con objetos de valor u objetos estáticos y sus

especializaciones. Posteriormente, haciendo uso de jQuery UI se generará la asociación entre entradas y valores y se desplegará la opción de autocompletado.

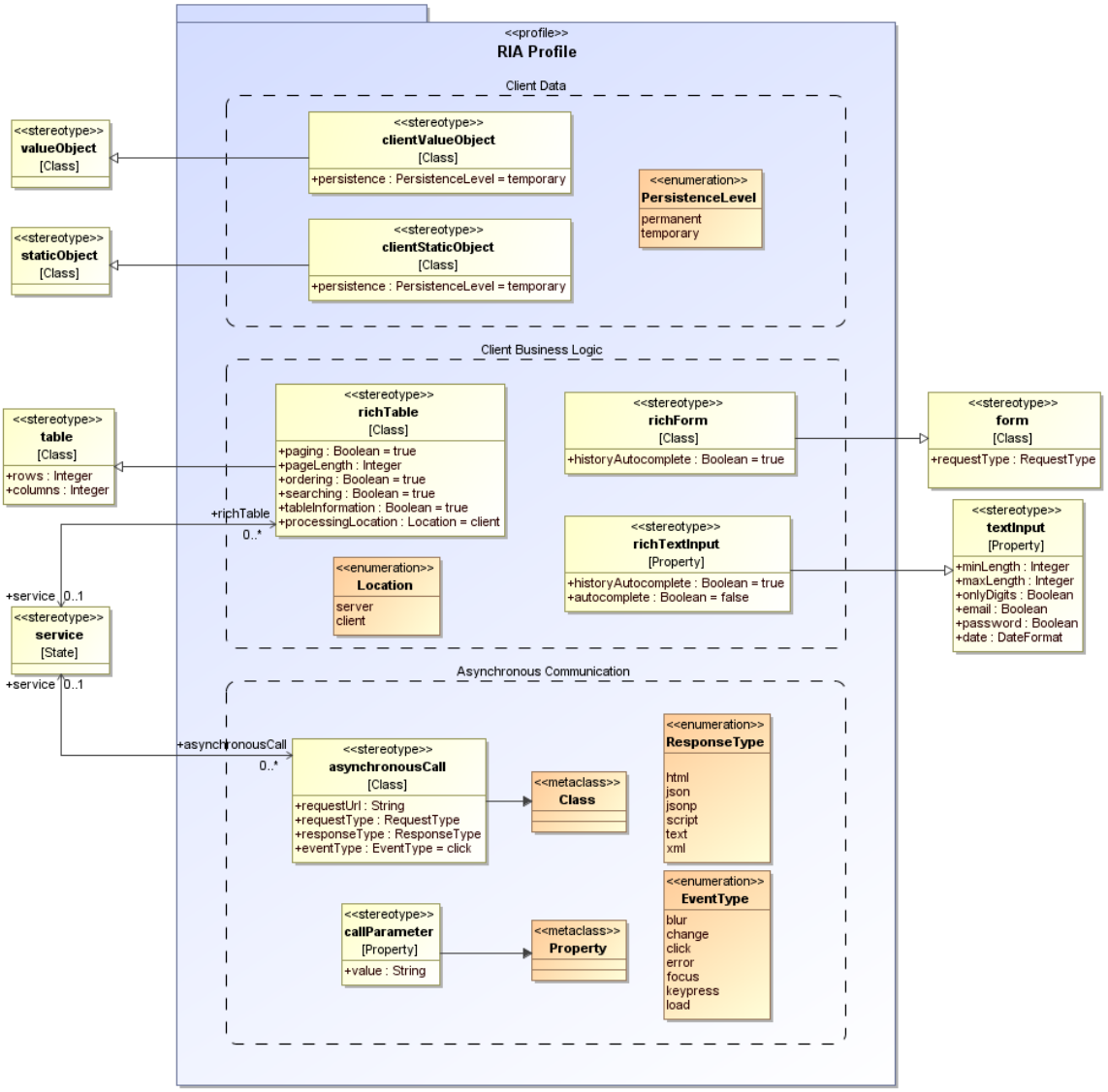

Figura 9: Perfil RIA.

La tabla enriquecida especializa a la clase tabla (*Table*) del diagrama de contenido. Permite especificar el servicio del diagrama de nodos encargado de poblar la tabla. Además, agrega nuevas propiedades a la tabla como permitir paginación (*paging*), especificar cantidad de registros por página (*pageLength*), permitir ordenamiento de columnas (*ordering*), buscar palabras en registros (*searching*), desplegar información de la tabla (*tableInformation*) y especificar dónde se procesarán estas funciones (*processingLocation*), ya sea en el servidor (server) o en el cliente (client). El plugin de jQuery, Datatables será utilizado para la implementación de una tabla enriquecida.

5.3. Comunicación asíncrona entre cliente y servidor

Para permitir comunicaciones asíncronas entre cliente y servidor se ha creado la clase denominada llamada asíncrona (*AsynchronousCall*). Esta clase es utilizada para realizar una solicitud AJAX (HTTP asíncrona) de un servicio.

La llamada asíncrona se asocia a uno o muchos elementos de interfaz de usuario del diagrama de contenido y a cero o un servicio del diagrama de nodos. Cuando ocurre un evento sobre algún elemento, se ejecuta un servicio. El servicio puede ser alguno especificado en el diagrama de nodos, o un servicio ejecutándose desde una URL ajena al sistema (por ejemplo un web service). La instancia de la llamada asíncrona puede ubicarse en el diagrama de contenido junto a sus elementos de interfaz de usuario asociados.

La clase posee un nombre (*name*), una URL (*requestUrl*) (utilizado en caso de que solicite un servicio externo), un tipo de solicitud (*requestType*) que funciona de igual manera al tipo de solicitud del formulario, un tipo de respuesta (*responseType*) que puede ser en formato *html*, *json*, *jsonp*, *script*, *text*, *xml* y un tipo de evento (*eventType*), que se aplica al elemento de interfaz de usuario y puede tomar los valores *blur*, *change*, *click*, *error*, *focus*, *keypress*, *load*. Además, la llamada asíncrona puede adjuntar ciertos parámetros a la solicitud, cada parámetro consiste en una propiedad de la clase y posee un nombre y un valor. Cada llamada asíncrona generará un método ajax() de jQuery.

6. Modelo de una RIA

A continuación se presenta un ejemplo de modelado de una aplicación haciendo uso de MoWebA con las extensiones RIA propuestas.

La aplicación consiste en un sistema de marcación de empleados en el cual un usuario invitado puede registrarse como empleado para posteriormente iniciar sesión y poder realizar marcaciones de entrada o salida. Si el usuario es un supervisor, es capaz de observar las marcaciones realizadas por los empleados.

Se sigue el proceso de modelado estipulado por MoWebA, el cual incluye la especificación del CIM, PIM, ASM y PSM de la aplicación sistematizado en sus distintas etapas con sus diagramas correspondientes.

Para facilitar la comprensión, solo se presentarán los diagramas del CIM/PIM y ASM/PSM que han sido extendidos por la propuesta y se destacarán los elementos recientemente agregados y/o que vayan a sufrir modificaciones al pasar del CIM/PIM al ASM/PSM.

6.1. CIM/PIM

Se han elaborado los distintos modelos y diagramas correspondientes al CIM (modelo de casos de uso) y PIM (modelos de entidades, navegacional, comportamiento, presentación, usuario) de MoWebA para el sistema de marcación de empleados.

El diagrama de entidades de la Figura 10 opera como un diagrama conceptual de base de datos y consta de dos entidades, *Empleados* y *Marcaciones* con una relación de uno muchos entre ellas. Se observa que las

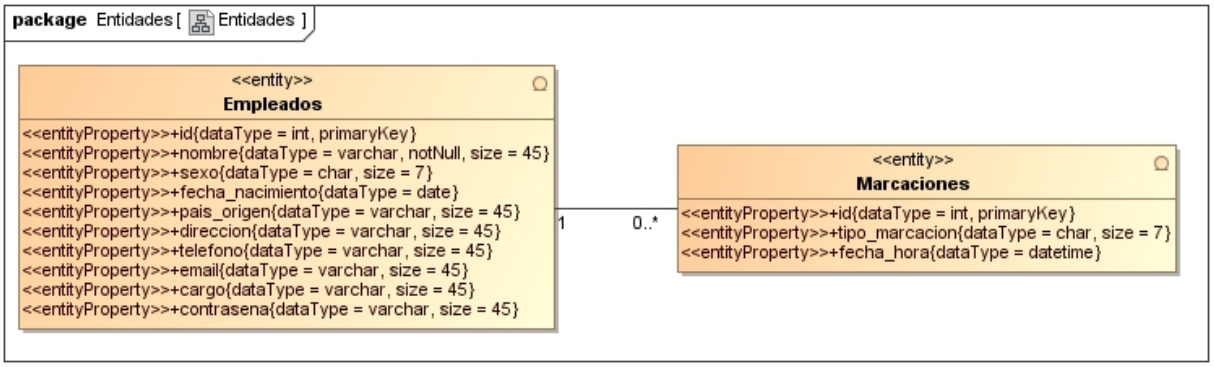

Figura 10: Diagrama de entidades del sistema de marcación de empleados.

propiedades de las entidades poseen el estereotipo <<*entityProperty*>> permitiendo agregar valores etiquetados para los tipos de datos *int*, *char*, *varchar*, *date* y *datetime*, tamaños máximos (7 y 45) y restricciones *primaryKey* y *notNull*.

La Figura 11 presenta el diagrama lógico de la aplicación. Se definen procesos de negocios encapsulados en las clases *ManejoSesiones*, *AlmacenamientoDatos* y *RecuperacionDatos*. También se definen los objetos de valor *EmpleadosRegistroVO* y *EmpleadosInicioVO* que dependen de la entidad *Empleados* y el objeto de valor *MarcacionesVO*. Se introducen cuatro objetos estáticos, *SexosSO*, *TipoMarcacionesSO*, *PaisesSO* y *CargosSO* definidas como clases con el estereotipo <<*staticObjetc*>> con propiedades representando valores. Las propiedades utilizan el estereotipo <<*value*>> y poseen valores etiquetados para el título de los valores (para desplegar en la interfaz) y para chequear el valor (para pre-seleccionar un valor en una lista).

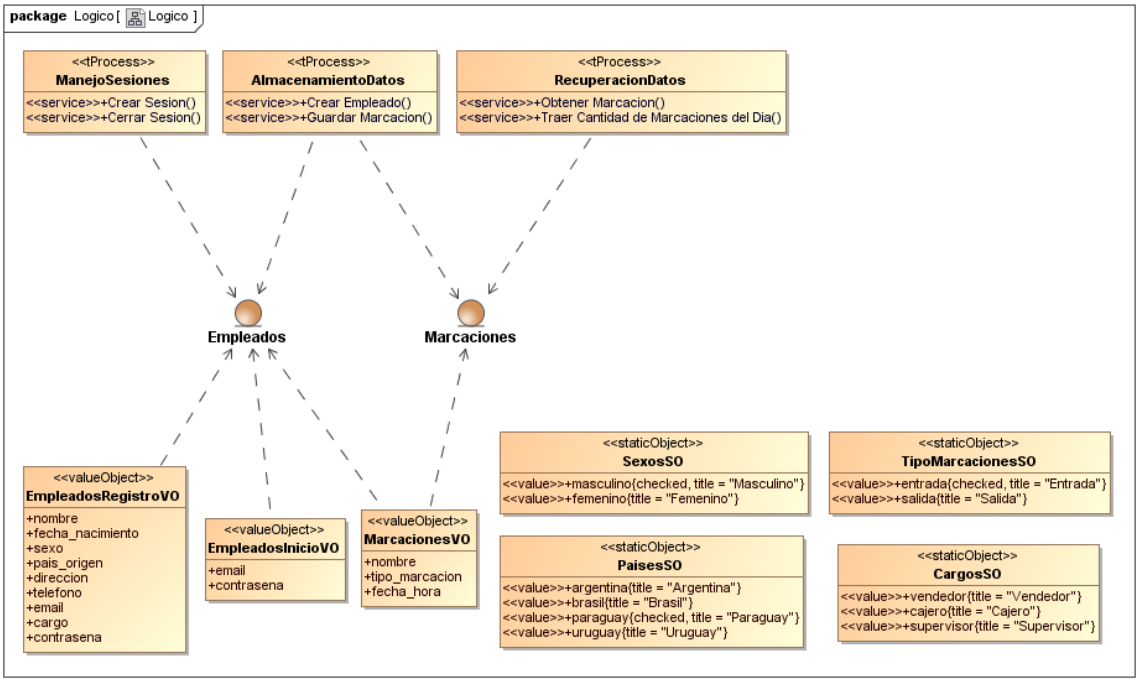

Figura 11: Diagrama lógico del sistema de marcación de empleados.

Se elaboraron diagramas de contenido para inicio de sesión, registro de empleados, realización de marcaciones y control de marcaciones. Los últimos tres serán explicados a continuación debido a que ellos abarcan la mayor cantidad de elementos de modelado que luego serán aprovechados para incluir funcionalidades RIA.

En la Figura 12 se observa el diagrama de contenido para el registro de empleados. Contiene la página de presentación *Registrar Empleado* compuesto por el formulario *RegistroEmpleado*. Este formulario posee valores etiquetados para el tipo de solicitud, que en este caso toma el valor insert/update. Además, el formulario posee entradas de texto con valores etiquetados para las restricciones como *mandatory*, *minLength*, *maxLength*, formato *date* (*default – dd/mm/yyyy*), *onlyDigits*, *email* y *password*. También se incluyen dos listas con valores etiquetados para los tipos *choice* y *dropbox*, éstas se asocian a los objetos estáticos definidos en el diagrama lógico para desplegar los valores de los objetos en la lista. Se agregó un enlace con un valor etiquetado que permite volver a la página de presentación *Iniciar Sesión* (que se encuentra relacionada con el estado virtual del diagrama de nodos *Iniciar Sesion*). Por último, se define un botón para el envío del formulario haciendo uso del estereotipo <<*submitButton*>>, como valor etiquetado se especifica el servicio encargado del procesamiento del formulario, en este caso el servicio del diagrama de nodos *Crear Empleado*.

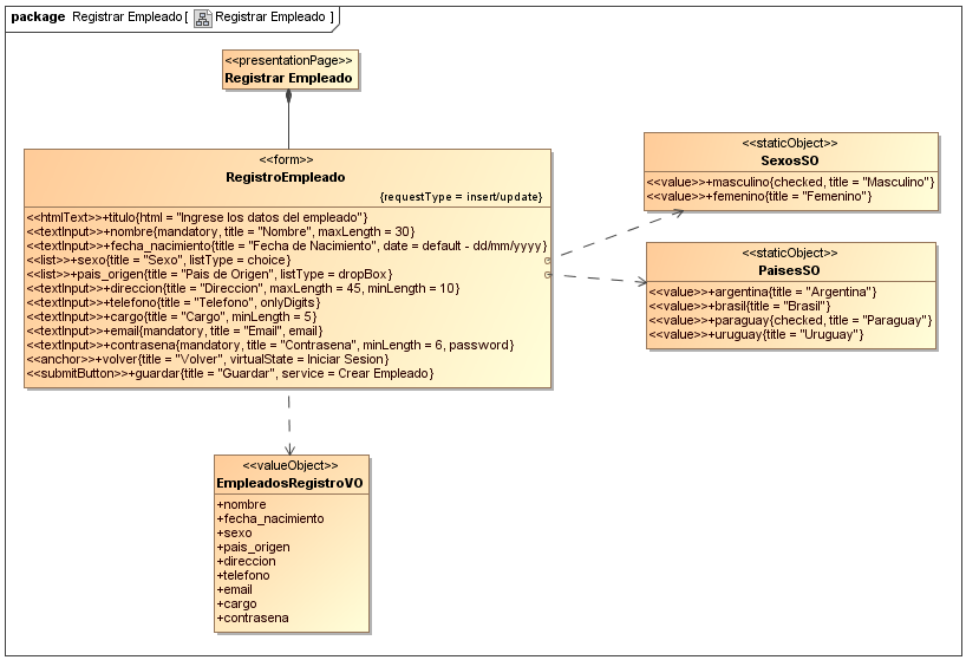

Figura 12: Diagrama de contenido para registro de empleados del sistema de marcación de empleados.

La Figura 13 contiene el diagrama de contenido para el control de marcaciones. Se dispone de la página de presentación *Controlar Marcaciones*. Esta página está compuesta por la tabla *Marcaciones* y el elemento compuesto de interfaz de usuario *CerrarSesion*. La tabla posee los encabezados declarados como texto HTML *nombre*, *tipo_marcacion* y *fecha_hora*. El elemento compuesto de interfaz de usuario contiene un enlace para cerrar la sesión de usuario, para ello se especifica como valor etiquetado el servicio del diagrama de nodos *Destruir Sesion* encargado de eliminar la sesión y de redirigir al usuario a donde sea necesario.

La Figura 14 presenta el diagrama de contenido para la realización de marcaciones. Posee la página de presentación *Realizar Marcación* que se encuentra a su vez compuesta por los elementos compuestos de interfaz de usuario *MarcacionesDelDia* y *CerrarSesion* y el formulario *IngresarMarcacion*. El elemento de

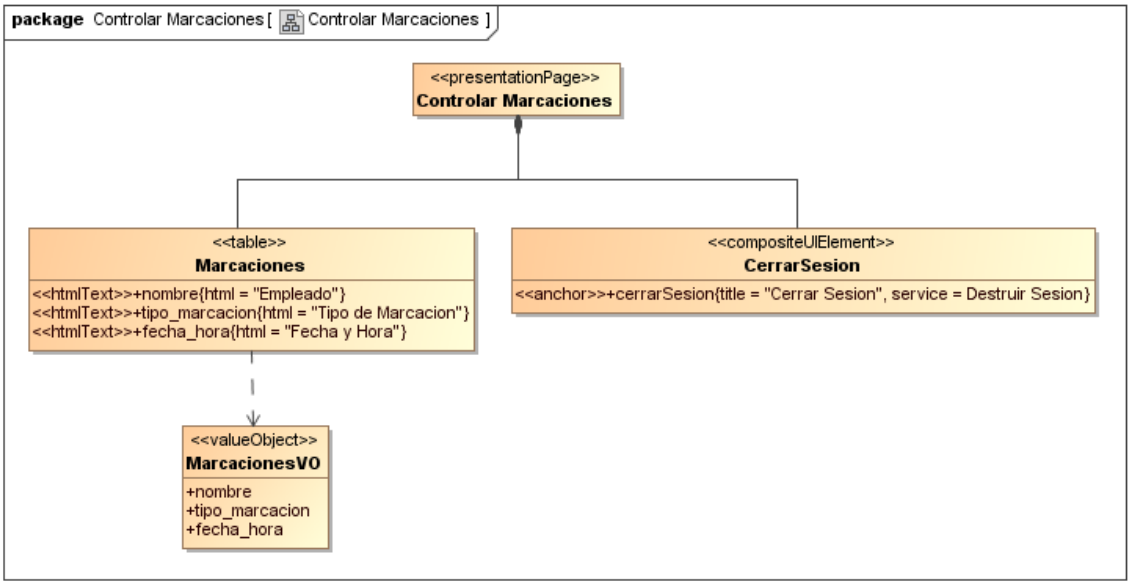

Figura 13: Diagrama de contenido para control de marcaciones del sistema de marcación de empleados.

interfaz de usuario *MarcacionesDelDia* contiene al enlace *actualizarMarcaciones*. Al presionar sobre este enlace se ejecuta el servicio *Traer Cantidad de Marcaciones del Dia*, que se encarga de actualizar el texto HTML *cantidadMarcaciones* de forma síncrona (con una recarga de la página entera). El formulario *IngresarMarcacion* y el elemento compuesto de interfaz de usuario *CerrarSesion* están definidos de la misma forma que en los diagramas de contenido anteriores.

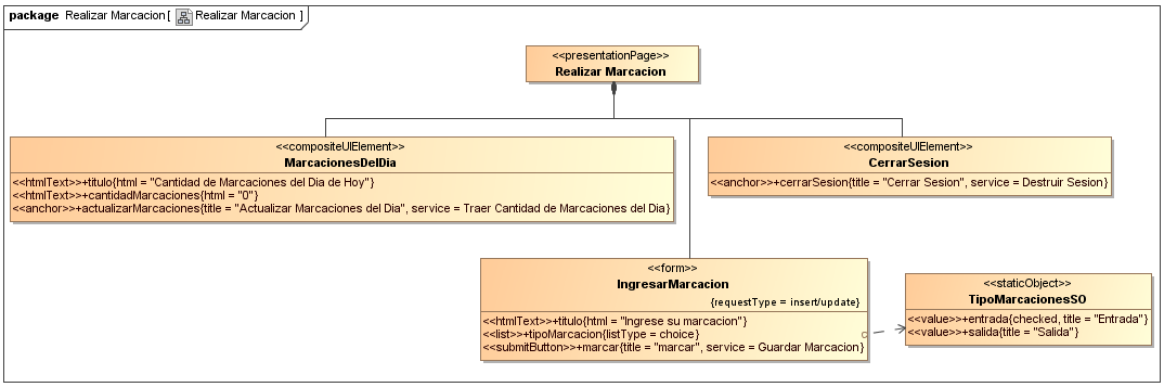

Figura 14: Diagrama de contenido para realización de marcaciones del sistema de marcación de empleados.

6.2. ASM/PSM

Posteriormente al modelado del CIM/PIM de la aplicación y siguiendo el proceso establecido por MoWebA, se procedió a hacer uso de las extensiones RIA propuestas para la definición de un ASM sobre el modelo elaborado. Cabe destacar que la propuesta no abarca la definición de un PSM, debido a que el ASM ya contempla información de plataforma específica para tecnologías basadas en scripting (Javascript, jQuery, jQuery UI, Datatables).

A continuación se presentarán las variaciones que han sufrido los modelos del PIM introducidos en la sección anterior para aprovechar las ventajas ofrecidas por las RIA. Se utiliza la misma convención de colores utilizada para las extensiones al PIM, el azul para las clases que han sido modificadas y el verde para las nuevas clases agregadas.

En primer lugar, se ha extendido el diagrama lógico de la aplicación con el fin de lograr almacenar datos en el cliente web (navegador). En la Figura 16 podemos observar dos variaciones con respecto al diagrama de la Figura 11. Las clases *PaisesSO* y *CargosSO* ahora poseen el estereotipo <<*clientStaticObject*>> y corresponden a objetos estáticos almacenados en el cliente. Ambos poseen valores etiquetados para indicar su nivel de persistencia. *PaisesSO* tiene un nivel de persistencia temporal, mientras que *CargosSO* tiene un nivel de persistencia permanente.

El diagrama de contenido para registro de empleados con extensiones para lógica de negocios en el cliente se observa en la Figura 15. A diferencia de la Figura 12, ahora el formulario corresponde a un formulario enriquecido gracias a la aplicación del estereotipo <<*richForm*>> sobre la clase. A partir de esto, se ha utilizado el valor etiquetado de autocompletado basado en historia para desactivar esta acción (si no se especifica nada, la opción de autocompletado basado en historia se encuentra habilitada por defecto en los formularios HTML). Las entradas de texto nombre y cargo ahora poseen el estereotipo <<*richTextInput*>>. La entrada nombre tiene el valor etiquetado para autocompletado basado en historia, permitiendo habilitar esta acción para la entrada a pesar de que se encuentre deshabilitada para el formulario. La entrada cargo posee el valor etiquetado de autocompletado permitiendo obtener los valores para autocompletado a través de la asociación de dependencia con el objeto estático en el cliente *CargosSO*. Por último, el elemento lista

país_origen ahora obtiene sus valores a desplegar del objeto estático en el cliente *PaisesSO* gracias a la relación de dependencia entre ellos.

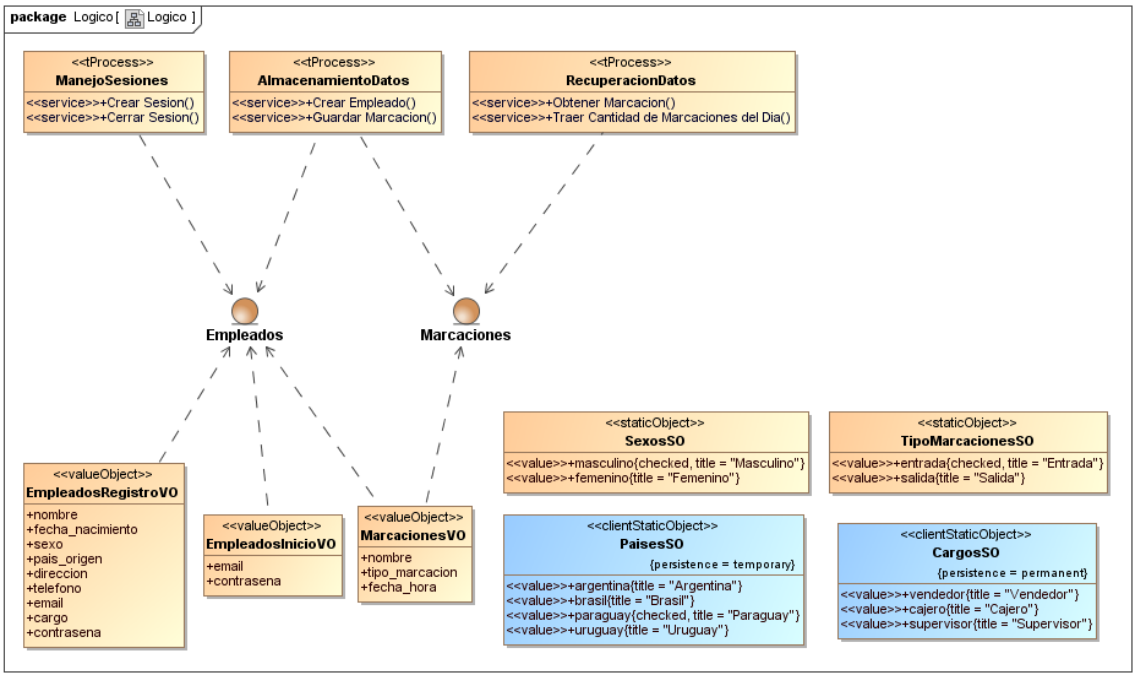

Figura 16: Diagrama lógico con extensiones RIA del sistema de marcación de empleados.

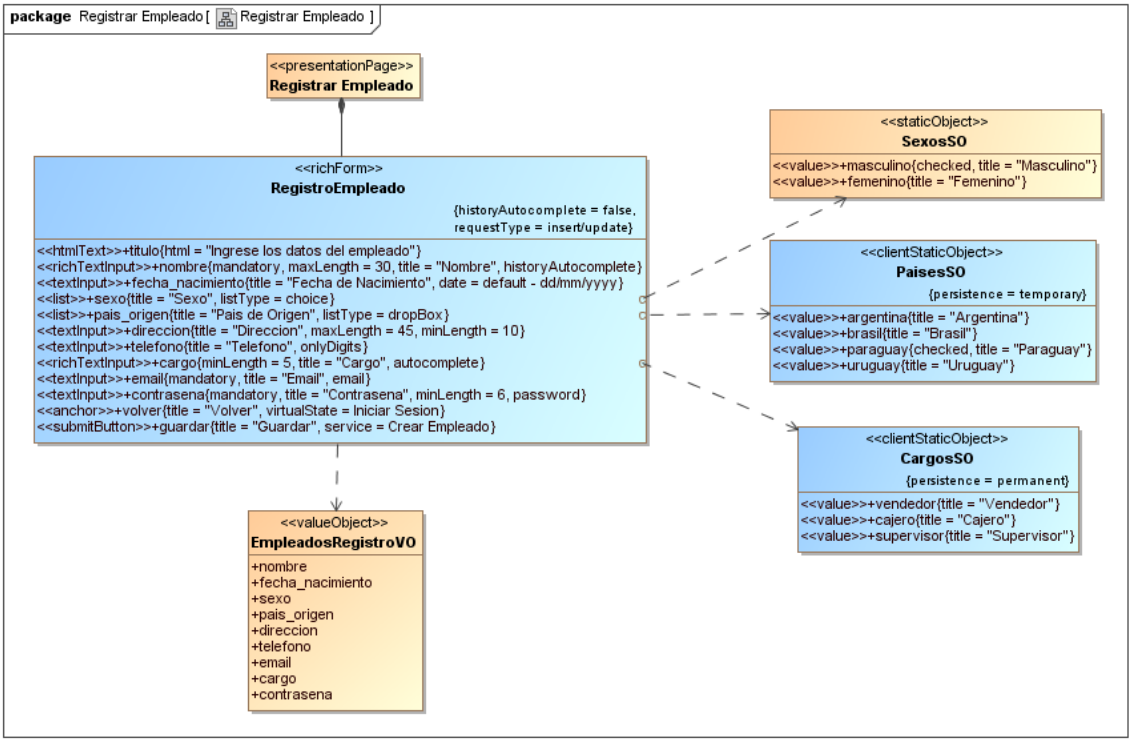

Figura 15: Diagrama de contenido para registro de empleados con extensiones RIA del sistema de marcación de empleados.

El diagrama de contenido para control de marcaciones extendido para RIA se puede apreciar en la Figura 17. La lógica de negocios en el cliente también es aprovechada, en este caso haciendo uso de la tabla enriquecida. En lugar de la clase del tipo tabla de la Figura 13, ahora se tiene una clase con estereotipo <<*richTable*>>, representando una tabla enriquecida con operaciones que pueden realizarse del lado del cliente o del servidor. Se utilizan los valores etiquetados para habilitar ordenamiento, paginación y búsqueda del lado del cliente. Además, se especifica el máximo de registros por página (15) y se habilita la información de la tabla. Por último, se fija el servicio *Obtener Marcaciones* del diagrama de nodos como encargado de proveer registros a la tabla.

La invocación asíncrona de servicios puede observarse en el diagrama de contenido para realización de marcaciones extendido de la Figura 18. Al diagrama de la Figura 14 se le agregan dos nuevas clases que representan llamadas asíncronas de servicios. La primera llamada es especificada a través de la clase *traerCantidadDeMarcacionesDelDia* con el estereotipo <<*asynchronousCall*>>. Esta clase se asocia mediante una relación de dependencia con el elemento de interfaz de usuario que lanza la llamada, en este caso el enlace *actualizarMarcaciones*. Por medio de valores etiquetados se especifican el evento que ocurre sobre el elemento de interfaz de usuario para lanzar la llamada (*click*), el tipo de solicitud (*retrieve*), el tipo de respuesta (*html*) y el servicio del diagrama de nodos que es llamado (*Traer Cantidad de Marcaciones del Dia*). Los parámetros de la llamada se definen como propiedades de la clase con el estereotipo <<*callParameter*>>. Esta llamada pasa como parámetro la variable de nombre *fechaActual* con valor *CURDATE()* definido como valor etiquetado. La segunda llamada asíncrona *guardarMarcacion* se asocia al botón de envío *marcar* del formulario enriquecido *IngresarMarcacion* y establece a través de valores etiquetados el tipo de evento sobre el botón de envío (*click*), el tipo de solicitud (*insert/update*), el tipo de respuesta (*html*) y el servicio a ejecutar (*Guardar Marcacion*). A diferencia de la llamada asíncrona anterior, ésta última no especifica ningún parámetro, sin embargo, como se encuentra relacionado con un formulario, los datos del formulario (tipo de marcación) serán enviados al servicio como parámetros.

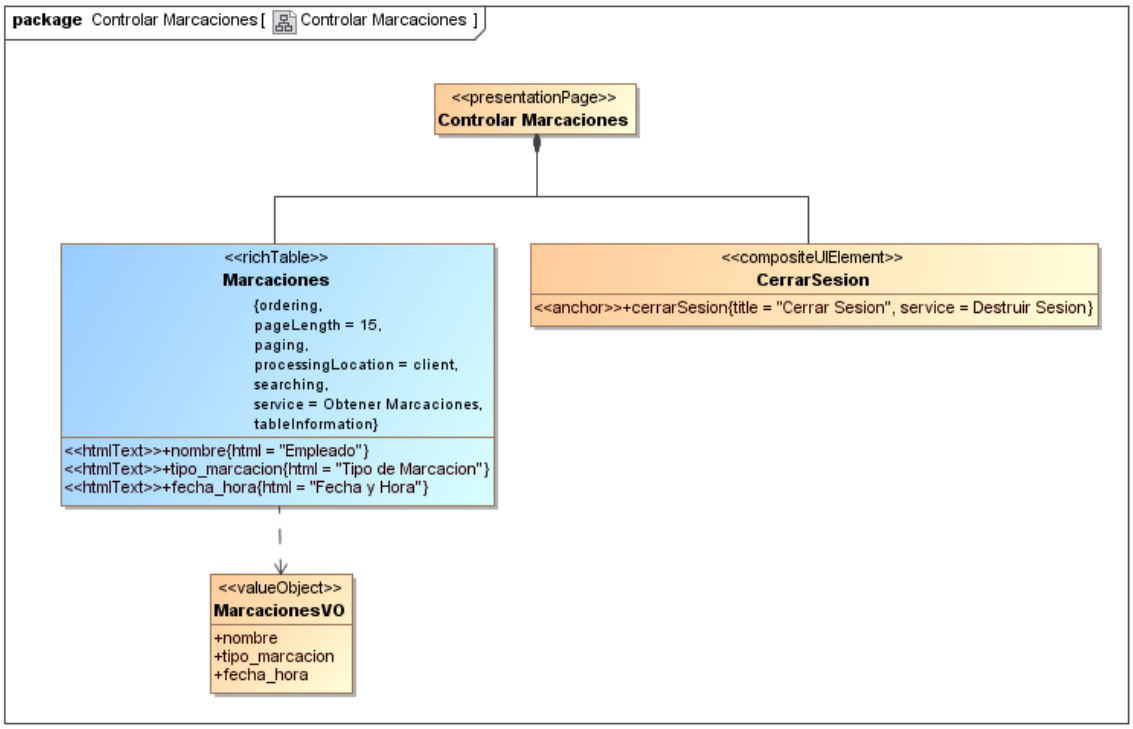

Figura 17: Diagrama de contenido para control de marcaciones con extensiones RIA del sistema de marcación de empleados.

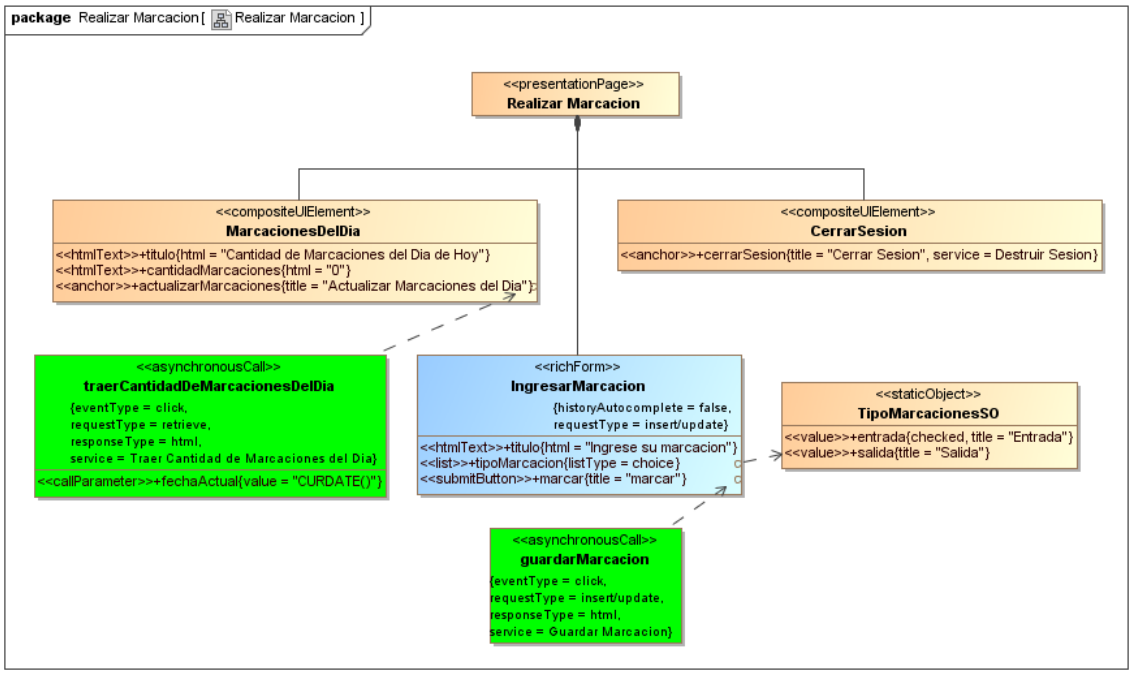

Figura 18: Diagrama de contenido para realización de marcaciones con extensiones RIA del sistema de marcación de empleados.

7. Referencias

[1] M. Brambilla, J. Cabot, and M. Wimmer, "Model-driven software engineering in practice," *Synthesis Lectures on Software Engineering*, vol. 1, no. 1, pp. 1–182, 2012.

[2] M. Brambilla, J. C. Preciado, M. Linaje, and F. Sanchez-Figueroa, "Business process-based conceptual design of rich internet applications," in *Web Engineering, 2008. ICWE'08. Eighth International Conference on*. IEEE, 2008, pp. 155–161.

[3] M. González, L. Cernuzzi, and O. Pastor, "A navigational role-centric model oriented web approach-moweba," *International Journal of Web Engineering and Technology*, vol. 11, no. 1, pp. 29–67, 2016.

[4] P. Fraternali, G. Rossi, and F. Sánchez-Figueroa, "Rich internet applications," *Internet Computing, IEEE*, vol. 14, no. 3, pp. 9–12, 2010.

[5] M. Busch and N. Koch, "Rich internet applications. state-of-the-art," *Ludwig-Maximilians-Universität München, München, Germany, Rep*, vol. 902, 2009.