Metodología de Modelado de Aplicaciones Web Móviles Basada en Componentes de Interfaz de Usuario

Pablo Vera[1], Claudia Pons[2], Carina Gonzáles[3], Daniel Giulianelli[1], Rocío Rodriguez[1]

[1] GIDFIS – Grupo de Investigación desarrollo y Formación en Innovación de Software – Universidad Nacional de La Matanza, Departamento de Ingeniería e Investigaciones Tecnológicas, Florencio Varela 1903, San Justo, Buenos Aires, Argentina
{pvera, dgiulian, rrodriguez}@ing.unlam.edu.ar
[2] LIFIA – Laboratorio de Investigación y Formación en Informática Avanzada – Universidad Nacional de La Plata, Calle 50 y 150, La Plata, Buenos Aires, Argentina
cpons@lifia.info.unlp.edu.ar
[3] Universidad de la Laguna, Departamento de Ingeniería de Sistemas y Automática y Arquitectura y Tecnología de Computadores. Área de Arquitectura y Tecnología de Computadores, La Laguna, España
cjgonza@ull.es

Resumen: Este trabajo presenta una metodología de modelado utilizando una extensión conservativa de UML que permite diseñar aplicaciones web móviles centrándose en el modelado de la interfaz de usuario mediante la utilización de componente configurables. El objetivo final es la generación automática del código fuente completo de la aplicación. Para lograr tal fin se extienden los diagramas de clases y componentes de UML. El diagrama de clases es extendido para poder generar la base de datos y sus relaciones. El diagrama de componentes es extendido mediante valores etiquetados que configuran cada componente permitiendo especificar su funcionalidad de forma clara y concisa pero a su vez conteniendo toda la información necesaria y suficiente para permitir generar una aplicación completa y funcional. Enmarcado en el ámbito de MDA esta metodología plantea distintas transformaciones, de modelo a modelo y de modelo a código basadas en el formato estándar para intercambio de diagramas UML denominado XMI.

Palabras Clave: MDA, UML, Modelado, Componentes, Móvil

Abstract. This paper shows a modeling methodology that uses a conservative extension of UML allowing designing mobile web applications focusing on the user interface modeling by using configurable components. The final goal is the automatic building of the complete source code of the application. In order to achieve this goal UML Class and Component diagrams were extended. The class diagram was extended to easiness the building of the database and it´s relationships. The component diagram was extended by tagged values that configure each component allowing defining it´s functionality in a clear and concisely way but also containing all necessary information to automatically build a complete and functional application. Framed within the scope of MDA,

this methodology proposes different transformations, from model to model and from model to code, all based on the standard format for UML model interchange called XMI.

Keywords: MDA, UML, Modeling, Components, Mobile

1 Introducción

El modelado de aplicaciones es un área muchas veces subestimada por la industria donde no se le da la suficiente importancia y muchas empresas, principalmente pequeñas o medianas, lo consideran una pérdida de tiempo. En otros casos sólo se utiliza en etapas tempranas del desarrollo para hacer una primera definición del problema y obtener los requerimientos. Gran parte de los modelos realizados en estas etapas luego no son actualizados con los cambios que surgen en la etapa de desarrollo haciendo que la documentación del sistema quede obsoleta. El modelado a bajo nivel de una aplicación también es una tarea ardua que agrega costos y tiempos que muchas empresas no están dispuestas a afrontar. Para subsanar todos estos problemas nace la arquitectura dirigida por modelos (MDA) [6], [10] donde los modelos van evolucionando y transformándose hasta llegar a generar el código fuente o parte del mismo en forma automática. Si bien actualmente existen herramientas basadas en esta arquitectura que generan código fuente, ninguna de ellas permite generar una aplicación completa 100% funcional, e incluso el esfuerzo necesario para realizar un modelado detallado que permita lograr dicha meta es demasiado elevado.

Por este motivo se plantea la necesidad de generar una nueva metodología de modelado que esté orientada a la generación del código fuente y que a su vez permita a los analistas y arquitectos definir de forma sencilla el comportamiento del sistema, sus pantallas y la información que el usuario deberá ver y manejar.

2 Uso de Componentes de Interfaz de Usuario

Una gran cantidad de aplicaciones se basan en la visualización y actualización de datos por parte del usuario de forma directa, sin requerir complejos procesos sobre los datos. Este tipo de aplicaciones pueden ser definidas especificando los datos que visualiza el usuario y cuales actualiza en cada parte del sistema. Al momento de especificar un sistema para ser programado es necesario definir que pantallas contendrá, que datos se visualizarán y cómo será la secuencia de navegación dentro del sistema. El objetivo buscado por esta metodología es lograr que esa especificación pueda realizarse en modelos UML de forma completa y que esos modelos puedan ser llevados luego a generar el código fuente completo de la aplicación ya que se brindarán todos los detalles necesarios en el modelo para lograr dicho objetivo.

Se plantea entonces la creación de una nueva metodología de modelado basada en componentes predefinidos de la interfaz de usuario parametrizables denominada CBHDM (Component Based Hypermedia Design Methodology – Metodología de Diseño Hipermedia Basada en Componentes). Estos componentes representarán

distintas pantallas comunes en los sistemas por ejemplo un listado de información, un menú, una pantalla de búsqueda, etc.

El primer paso será construir el modelo de dominio donde se definen las clases que servirán luego para configurar los componentes y establecer la forma de visualización y actualización de los objetos dentro del sistema. Luego se crea el modelo de Navegación e Interfaz que permite combinar los distintos componentes de la interfaz de usuario y configurarlos definiendo la visualización, actualización de datos y la navegación dentro de la aplicación.

La metodología está basada en UML. Por lo tanto se utiliza una extensión del lenguaje para definir ciertas características necesarias sobre los diagramas de clases y componentes. Esta extensión del lenguaje está definida mediante el perfil UML de la figura 1. Las siguientes secciones explican cómo se utilizan las extensiones del lenguaje para modelar una aplicación utilizando esta metodología.

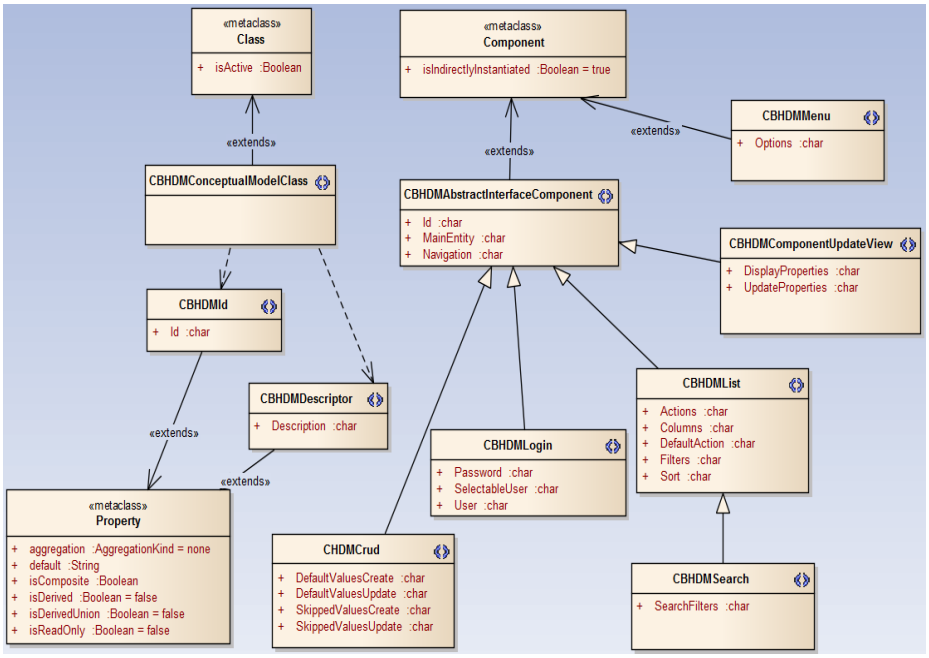


Fig. 1. Perfil UML de la metodología CBHDM

3 Modelado de Aplicaciones Web Móviles

La metodología se plantea para el modelado de aplicaciones web móviles por las siguientes razones:

- **Interfaz reducida**: Las pantallas de los dispositivos móviles son pequeñas, más allá de la resolución que posean el tamaño es reducido, y por lo tanto la interfaz presentada al usuario debe ser simple, cómoda y adecuarse al método de

utilización. Como esta metodología está basada en componentes, cada uno de ellos va a representar una pantalla que se mostrará al usuario. Y al ser, las interfaces de usuario, necesariamente simples, hacen que la configuración de los componentes sea directa sin necesidad de especificar distribuciones complejas de controles. La configuración se centra en la información básica a visualizar con una distribución estándar optimizada para la visualización en dispositivos móviles.

- **Sistema de Navegación simple e intuitivo**: Al tener una pantalla reducida el sistema de navegación también debe minimizarse. Siguiendo las pautas del W3C (World Wide Web Consortium) sobre sitios web móviles [15], se incorpora a cada componente una barra de navegación reducida que será mostrada en la parte superior de la pantalla.
- **Aprovechamiento de características especiales**: Se incorporan ciertos elementos que permiten aprovechar las características de algunos dispositivos móviles. Por ejemplo el uso de la geolocalización mediante el GPS del dispositivo si está presente o la posibilidad de incorporar links especiales para realizar llamadas o enviar SMS si se está mostrando un número telefónico.

Sin embargo esta metodología puede aplicarse también a aplicaciones web tradicionales donde no se requieran layouts de pantallas demasiados complejos.

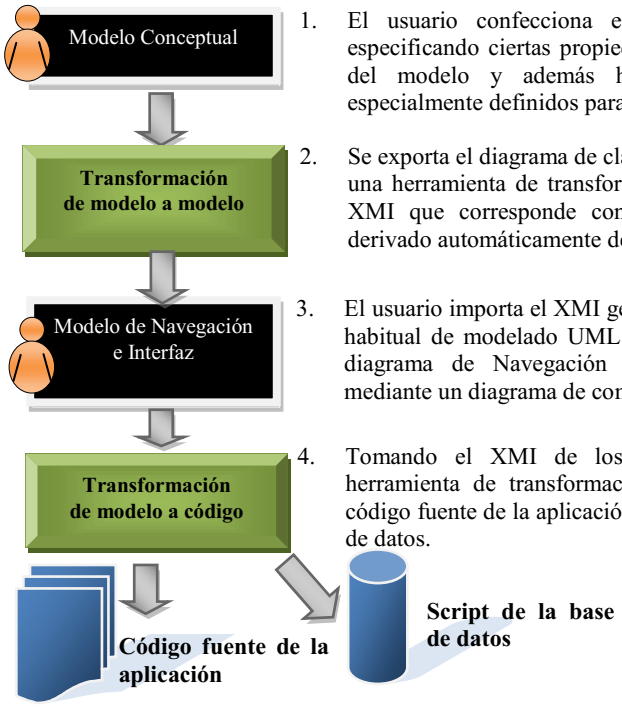
4 Pasos de la Metodología

El modelado de sistemas hipermedia es una práctica que data de varios años atrás cuando la web comenzaba a popularizarse. Uno de los trabajos más relevantes del área es OOHDM (Object Oriented Hypermedia Design Method) [13] que a su vez está basado en un trabajo previo llamado HDM (Hypermedia Design Method) [3].

OOHDM establece cuatro actividades principales para el diseño hipermedia. Estas actividades son: modelado conceptual, modelado de la navegación, diseño de la interfaz abstracta e implementación.

La metodología propuesta respeta estas actividades pero unifica el modelado de la navegación y de la interfaz abstracta en un único diagrama.

En la figura 2 se muestran los pasos de la metodología, detallando las acciones del usuario y las transformaciones automáticas realizadas.



1. El usuario confecciona el diagrama de clases UML, especificando ciertas propiedades definidas en la extensión del modelo y además hace uso de tipos de dato especialmente definidos para aplicaciones móviles.
2. Se exporta el diagrama de clases al formato XMI y mediante una herramienta de transformación se obtiene otro archivo XMI que corresponde con un diagrama de navegación derivado automáticamente del diagrama de clases.
3. El usuario importa el XMI generado y utiliza su herramienta habitual de modelado UML para modificar y completar el diagrama de Navegación e Interfaz que es realizado mediante un diagrama de componentes UML extendido
4. Tomando el XMI de los dos modelos realizados, la herramienta de transformación genera como resultado el código fuente de la aplicación y el script para crear las bases de datos.

Fig. 2. Pasos de la metodología

A continuación se detallan cada uno de los modelos con sus diagramas y extensiones.

4.1 Modelo Conceptual

El modelo conceptual del sistema se realiza utilizando el diagrama de clases UML donde se especificarán las distintas entidades. El modelado se realiza con clases aisladas y las relaciones se especifican en la propia clase creando propiedades del tipo de la clase relacionada.

El diagrama de clases es extendido mediante:

- **Estereotipos para propiedades especiales**: a fin de facilitar la generación del script de la base de datos a partir de este modelo, es necesario identificar de cada clase la propiedad correspondiente al identificador único del objeto y la propiedad que contenga una descripción legible por el usuario. La primera se transformará en la clave primaria de la tabla correspondiente a dicha entidad y la segunda se utilizará en la interfaz de usuario cuando se muestre un objeto de dicha clase, por ejemplo en un combo de selección o un listado.

- **Tipos de dato para móviles**: para aprovechar las características propias de los dispositivos móviles se definen dos nuevos tipos de datos:

  (a) `phoneNumber`: al identificar una propiedad como número telefónico será posible que se genere un link en la interfaz de usuario para hacer una llamada en forma directa a dicho número o enviar un SMS.

  (b) `address`: el campo dirección va a permitir utilizar las características de geolocalización presente en una gran gama de dispositivos móviles permitiendo por ejemplo que con un click pueda ubicar dicha dirección en un mapa o crear una ruta para navegar con el GPS. Si el dispositivo no dispone de GPS entonces el usuario verá igualmente la dirección pero en forma de texto plano.

También en el modelo conceptual se hará uso de las clases del tipo enumeration (estereotipo ya existente en UML) para especificar por ejemplo estados de un determinado objeto.

La figura 3 muestra un ejemplo de un diagrama de clases con las extensiones mencionadas.

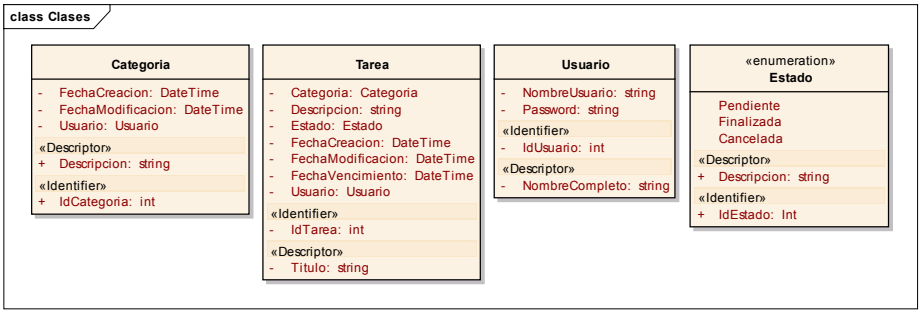


Fig. 3**.** Ejemplo de Modelo Conceptual de una aplicación

4.2 Modelo de Navegación e Interfaz

Para poder modelar la interfaz y la navegación a través del sistema se utiliza el diagrama de componentes de UML. Cada componente se transformará luego en una página web, es por eso que los componentes están definidos de forma simple para que sean adecuados para un dispositivo móvil e incorporan también la navegación necesaria para moverse dentro de la aplicación.

Se crearon distintos estereotipos para poder identificar cada uno de los tipos de componentes. A su vez cada tipo de componente posee un conjunto de valores etiquetados los que permiten configurarlos para adaptarse a las necesidades de cada aplicación. Existen algunos valores etiquetados comunes para todos los componentes, ellos son:

- `id`: identificador de cada componente y se utilizará para poder referenciar un componente desde otro para la navegación.
- `title`: texto que se mostrará en la pantalla final mostrada al usuario permitiendo identificar la misma.
- `navigation`: incluye los links que aseguran la navegación entre los componentes del sistema.
- `MainEntity`: valor etiquetado que está presente en la mayoría de los componentes y se refiere a la clase del modelo conceptual en el cual dicho componente está basado. Por ejemplo, en un listado si bien se incorpora información de distintas clases siempre existirá una clase que será la base de la consulta y a partir de ella se accederán a las clases relacionadas.

Los componentes definidos son `Login`, `List`, `Search`, `Menu`, `CRUD`, y `UpdateView`.

- `Login`; este componente permite autentificar un usuario en el sistema. La figura 4 muestra el componente con sus valores etiquetados. `User` y `Password` se refieren a las propiedades de la clase contra las cuales se realizará la autenticación. `RedirectToComponent` indica el componente al cual el usuario será redireccionado tras un logueo exitoso. Para reducir el ingreso de información en la aplicación móvil se agrega la posibilidad de configurar el componte para que el usuario en lugar de ser ingresado en forma textual sea seleccionado mediante un combo. Esta característica se configura mediante el valor etiquetado `SelectableUser`.

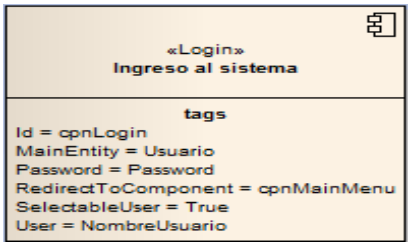


Fig. 4. Componente del tipo Login

- `List`: listado de información que puede visualizarse como una grilla o tabla con filas y columnas que muestren información de las entidades. En este punto es importante considerar el tamaño reducido de las pantallas de los dispositivos móviles ya que no permitirá visualizar un listado con muchas columnas. Se recomienda diseñar la aplicación con una columna principal de información y alguna columna adicional de pequeño tamaño como por ejemplo una fecha o un estado. Para poder incorporar información adicional se crea un valor etiquetado denominado `AditionalInformationLine` que permitirá mostrar una

segunda línea de información en cada fila. Esta es una práctica muy utilizada en las interfaces de los diferentes dispositivos móviles como puede verse en las figura 5.

Fig. 5**.** Ejemplos de listados de dispositivos móviles donde puede apreciarse la presencia de una columna principal de información, columnas secundarias de tamaño pequeño y una fila de información adicional.

También pensando en los dispositivos móviles se facilita la configuración de las acciones al hacer click sobre cada una de las filas del listado. Se incorpora una acción por defecto al clickear denominada `DefaultAction` que incluye un link a otro componente al cual pueden pasarse parámetros. También el valor etiquetado `Actions` incorpora las distintas opciones sobre el listado que serán mostradas, con un menú al usuario tras una presión larga si se trata de una pantalla táctil, generando una interfaz similar a la utilizada en el sistema operativo Android o IOS como puede verse en la figura 6. En el caso de que `DefaultAction` no sea especificado el click sobre una fila de la grilla mostrará por defecto el menú de acciones especificado en el tag `Actions`.

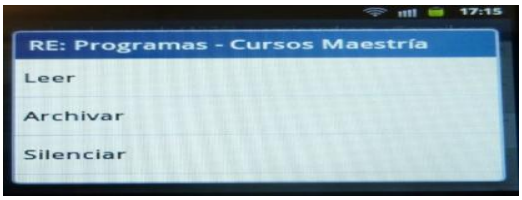


Fig. 6**.** Ejemplo de menús secundarios desplegados luego de una presión larga en pantallas táctiles.

La tabla 1 muestra un ejemplo de los valores etiquetados de un componente del tipo `List` donde puede apreciarse que la lista es totalmente configurable permitiendo entre otras acciones ordenar los datos y predefinir los filtros con los que el usuario visualizará dicho listado, en el ejemplo, el listado muestra viajes del usuario logueado que cumplan con un cierto estado. También puede apreciarse que el contenido de las columnas puede obtenerse mediante una función que calcule un dato del sistema. En el ejemplo la segunda columna recupera la fecha de solicitud del viaje de una tabla auxiliar de log relacionada con el viaje.

Tabla 1. Ejemplo de los valores etiquetados de un componente List

Tag	Value
Id	cpnCurrentTrips
Navigation	Link("Main Menu", cpnMainMenu,,0)
MainEntity	Viaje
FixedFilters	Driver = LOGGEDUSER AND TripStatus in (TripStatus.Accepted, TripStatus.Started)
Columns	DestinationAddress; Retrieve(min(EventDateTime), TripLog, TripStatus=TripStatus.Pending, "Request Date");
Actions	OptionalLink("Start", cpnStartTrip, "ObjectID = TripID",TripStatus=Accepted AND not Exist (TripStatus.Started)); OptionalLink("Finish", cpnFinishTrip, "ObjectID = TripID", TripStatus=Started); OptionalLink("Decline", cpnDeclineTrip, "ObjectID = TripID",TripStatus=Accepted);
Sort	"Request Date" ASC

- `Search`: es similar a List pero incorpora la posibilidad de agregar Filtros que serán ingresados por el usuario para acotar la búsqueda. Agrega un único valor etiquetado denominado `SearchFilters` que permite especificar: la propiedad por la cual buscar y el tipo de filtro a aplicar. Entre los tipos de filtros disponibles están: `SingleSelection`, `MultipleSelection`, `FreeText`, `BooleanType y DateRange`.

- `Menu`: define un menú con links a otros componentes. Dentro del valor etiquetado `Options` se configuran los distintos links que serán las opciones del menú. La tabla 2 muestra ejemplos de valores etiquetados del componente `Menu`, donde puede apreciarse que además de especificarse el componente destino del link puede pasarse parámetros a los mismos.

Tabla 2. Ejemplo de valores etiquetados del componente Menu

Etiqueta	Valor
Id	cpnMainMenu
Navigation	Link("Logout", cpnLogin,,)

	Link("Ver tareas Pendientes", cpnPendingTasks,,);
Options	Link("Agregar tarea", cpnTask, action=C,); Link("Buscar Tarea", cpnTaskSearch,,); Link("Categorias", cpnLstCategories,,)

- `CRUD:` es el encargado de mostrar, actualizar, crear o eliminar un objeto. Permite definir valores por defecto para algunas propiedades al crear o actualizar y además permite omitir modificar ciertas propiedades según si se crea o actualiza el objeto. La tabla 3 muestra un ejemplo de los valores etiquetados del componente con distinta configuración de propiedades para la creación y actualización.

Tabla 3. Ejemplo de valores etiquetados del componente CRUD

Etiqueta	Valor
Id	cpnTask
Navigation	Link("Menu Principal", cpnMainMenu,,); Link ("Atras", BACK,,)
MainEntity	Tarea
DefaultValuesCreate	Estado=Estado.Pendiente; Usuario=LOGGEDUSER; FechaCreacion=NOW
DefaultValuesUpdate	FechaModificacion=NOW
SkippedPropertiesCreate	FechaModificacion
SkippedPropertiesUpdate	FechaCreacion; Usuario

- `UpdateView:` operación de actualización con características especiales. Es posible que la actualización de un objeto se realiza por partes, donde algunas de sus propiedades se actualizan y otras no. Es por eso que este componente permite:
  - mostrar algunas propiedades sin modificarlas mediante el tag `DisplayProperties`
  - definir puntualmente cuales son las propiedades a actualizar mediante el tag `UpdateProperties`
  - crear entidades relacionadas como en el ejemplo de la tabla 4 se crea un registro en la tabla log al grabar la entidad principal del viaje.

Tabla 4. Ejemplo de valores etiquetados del component UpdateView

Etiqueta	Valor
Id	cpnAcceptTrip
Navigation	Link("Main Menu", cpnMainMenu,,0); Link("Back", cpnPendingTrips,,9);
MainEntity	Trip
CreateEntity	TripLog
DisplayProperties	CustomerAddress; DestinationAddress;

	`Retrieve(min(EventDateTime),TripLog, TripStatus=TripStatus.Pending,"Request Date");`
	
	`CustomerPhone`

`UpdateProperties`	`TripLog.Remarks`

`DefaultValuesUpdate`	`TripStatus = TripStatus.Accepted;`
	`Driver = LOGGEDUSER;`
	`TripLog.Driver = LOGGEDUSER;`
	`TripLog.EventDateTime = NOW`

5 Transformaciones

Esta metodología se enmarca en el enfoque MDA (Arquitectura Definida por Modelos) donde todo el desarrollo está basado en la realización de modelos y mediante transformaciones se va avanzando hasta llegar a generar código fuente. Tal como se mostró en los pasos de la Figura 1 esta metodología incluye dos transformaciones, la primera de ella de modelo a modelo y la segunda de modelo a código. A continuación se detallan cada una de ellas.

5.1 Transformación de Modelo a Modelo

Una vez finalizado el modelo conceptual el usuario exporta el diagrama de clases a un archivo XMI [12] y lo importa en la herramienta de transformación. La herramienta reconoce cada una de las clases y genera un nuevo archivo XMI correspondiente a una primer versión del diagrama de navegación e interfaz que luego el usuario podrá modificar y completar según las necesidades de la aplicación. De forma automática se generan las siguientes transformaciones:

1. Por cada clase del modelo conceptual se genera un componente del tipo `CRUD` que va a permitir administrar los objetos de dicha clase. Se exceptúan las clases del tipo `enumeration`.
2. Por cada clase del modelo conceptual se genera un componente del tipo `List` que va a permitir visualizar el listado de objetos de dicha clase y se generan acciones con links hacia el componente `CRUD` creado en el punto 1. Se exceptúan las clases del tipo `enumeration`.
3. Se genera un componente del tipo `Menu` con links a los distintos componentes del tipo `List` creado en el punto 2.

De esta forma se obtiene un diagrama de componentes donde de forma automática ya se dispone de un menú principal, listados y edición de cada una de las entidades del sistema que deben ser administradas, es por eso que se excluyen las clases del tipo `enumeration` ya que los valores de las mismas no son modificados por el usuario sino que son definidos en el mismo modelo.

Esta es una transformación de Modelo a Modelo que se implementa a través de una transformación sobre la representación textual de dichos modelos (XMI).

5.2 Transformaciones de Modelo a Código

Para obtener la aplicación funcional se deben realizar dos transformaciones, la primera tomará el modelo conceptual y generará el script de la base de datos. Se realiza mediante una transformación de Modelo a Texto M2T tomando nuevamente el XMI como representación del modelo.

La segunda y última transformación tiene como entrada los archivos XMI del modelo conceptual y del modelo de navegación e interfaz. En este punto la transformación es más compleja y se debe realizar además una validación para verificar que los valores etiquetados hayan sido definidos según las reglas establecidas.

Para establecer las normas de escritura de los distintos valores etiquetados de cada componente se genera un lenguaje que define todo el modelo de configuración. Dicho lenguaje está expresado en la forma BNF (Backus Naur Form) [1].

Este BNF fue generado incluyendo todas las características de los distintos componentes del sistema y además incluye una sección que debe ser generada en forma dinámica por la herramienta de transformación para poder validar la correcta escritura de entidades y propiedades de las mismas dentro del modelo. La figura 7 muestra un ejemplo de un código BNF que deberá ser generado en forma automática por la herramienta para el caso de un sistema de administración de viajes en taxi.

```
135  <Entity> ::= 'Trip' | 'TripLog'  | 'Administrador' | 'Driver'
136
137  <EntityProperty> ::=  <TripEntityProperty> | <TripLogEntityProperty> | <Admin
138  <TripLogEntityProperty> ::= 'TripLog.'<TripLogProperty> | 'TripLog.'<DriverEn
139  <TripEntityProperty> ::= 'Trip.'<TripProperty> | 'Trip.'<DriverEntityProperty
140  <DriverEntityProperty> ::= 'Driver.'<DriverProperty>
141  <TripStatusEntityProperty> ::= 'TripStatus.'<TripStatusProperty> | 'TripStatu
142  <AdministratorEntityProperty> ::= 'Administrator.'<AdministratorProperty>
143
144  <DriverProperty>::='Password' | 'UserName' | 'DriverID' | 'Name'
145  <TripProperty> ::= 'CustomerAddress' | 'CustomerName' | 'CustomerPhone'
146  <TripLogProperty> ::= 'EventDateTime' | 'Remarks' | 'TripLogID'
147  <TripStatusProperty> ::=  'Description' | 'TripStatusID'
148  <TripStatusEnumValues> ::=  'Pending' | 'Accepted' | 'Started' | 'Finished' |
149  <AdministratorProperty> ::= 'Password' | 'UserName' | 'Name' | 'UserID'
150
151  !se genera en forma dinamica segun los componentes definidos en el XMI
152  <ComponentId> ::= 'cpnLogin' | 'cpnMainMenu' | 'cpnCurrentTrips' | 'cpnEditTr
153
```

Fig. 7. Fragmento de código BNF generado en forma dinámica para una aplicación tomando como base el modelo conceptual

Para realizar esta transformación será entonces necesario transformar el diagrama de componentes parametrizado con los valores etiquetados a un código fuente escrito en el lenguaje definido por el BNF. Luego se debe realizar un compilador que interprete dicho lenguaje y genere el código fuente en el lenguaje destino deseado.

Algunas de las actividades realizadas durante la generación del código fuentes son las siguientes:

- Cada propiedad pública de una clase será mapeada a una propiedad en la clase en el lenguaje de programación destino mediante una variable privada y su correspondiente acceso público.
- Según la navegación se crean las diferentes páginas y vistas con el acceso a datos.
- Para las pantallas de actualización de datos (CRUD) se derivan los controles de forma automática de acuerdo al tipo de dato, si la relación es con otra entidad se pone un combo de selección con la descripción de la entidad relacionada.
- Cada uno de los componentes especificados en el modelo de navegación deberá ser renderizado a un control según su funcionalidad.
- Para las clases del tipo enumeración se creará un tipo de clases de negocio particular donde no tendrá actualización pero si acceso a los datos. A su vez se deberán generar los valores posibles de dicha enumeración en el script de base de datos adicionando los registros correspondientes.

La aplicación web será generada teniendo en cuenta las buenas prácticas para aplicaciones web móviles de la W3C [16] lo que asegura entre otras cosas optimizar el uso de la red y generar una buena experiencia de usuario.

6 Trabajos Relacionados

Existen diversos trabajos que hacen uso de componentes en la arquitectura MDA. Cada uno de ellos aborda el tema con un enfoque particular. "An MDA approach to tame component based software development" [4] incorpora los componentes pero centrándose en sus interfaces para especificar la interconexión de los mismos para un sistema particular. "Foundations of Component-Based Development and MDA" [2] incorpora un lenguaje propio llamado NEREUS que permite parametrizar componentes, este lenguaje luego es traducido a OCL para adaptarse al estándar de UML. La parametrización que proponen es diferente a la planteada en este trabajo ya que se enfoca al componente en función del metamodelo y no para configurar su comportamiento. "A Domain-Specific Modeling Approach For Component-Based Software Development." [17] y "Towards A Multiviews Component Based Model Driven Approach" [5] utilizan componentes en MDA pero se refieren a componentes del sistema a nivel general de objetos y no componentes de la interfaz de usuario, es decir no crea tipos pre-definidos de componentes y tampoco permiten configurarlos. De forma similiar "A Model Driven Component Framework for Mobile Computing" [14] define una serie de componentes en el ámbito de MDA pero basados en el estándar CORBA [9] permitiendo definir de forma rápida los medios de comunicación entre distintos sistemas.

Otro trabajo, que si bien no incorpora la utilización de componentes, es interesante como metodología de desarrollo web basado en MDA es "Towards Interoperable Web Engineering Methods" [8] que plantea un metamodelo de referencia con 13

modelos que incluyen la lógica de negocios, Modelo de datos e Interfaz de Usuario. También incluye un profile para la interfaz donde no se modela un componente completo sino que cada uno de sus partes por separado es decir botones, texto, controles de formulario, etc. lo que hace más trabajosa la tarea de modelar una pantalla estándar.

Existen también herramientas específicas para generar aplicaciones móviles sin escribir código. Un claro ejemplo de ello es MIT App Inventor [7] que reemplaza la programación tradicional por una programación enteramente gráfica pero no lo hace a nivel de un modelo general sino que directamente reemplaza las líneas de código fuente por representaciones gráficas de las mismas que se van concatenando. Por ejemplo un gráfico representa un evento y dentro es posible arrastrar otro gráfico que representa una acción determinada. Esta herramienta sólo permite realizar aplicaciones nativas para sistemas operativos android.

7 Conclusiones y Trabajos Futuros

El modelar un sistema no debe ser considerado como un esfuerzo extra que se suma al desarrollo del mismo. El presente artículo muestra que es posible utilizar los modelos realizados para generar el código fuente de una aplicación, asegurando además, una documentación actualizada y una trazabilidad real entre los modelos y el código de la aplicación.

La metodología planteada permite modelar aplicaciones web móviles de una forma rápida y sencilla asegurando que los modelos creados contengan toda la información necesaria para poder derivar el código fuente y las bases de datos de la aplicación de forma totalmente automática.

Todos los diagramas empleados están basados en UML con extensiones propias del lenguaje lo que permite al diseñador poder usar cualquier herramienta de modelado siempre que cumpla con el estándar de intercambio XMI, que es el formato utilizado como base para realizar las transformaciones.

Si bien los componentes incorporan características especiales para diseñar aplicaciones web móviles es posible reutilizar esta metodología para realizar aplicaciones web tradicionales que administren datos.

En muchos casos las aplicaciones móviles tienen su contrapartida en la web tradicional donde generalmente se realiza la carga masiva de datos, es decir se dispone de un sistema backend de administración y el frontend de la aplicación es el sistema móvil. Esta metodología permite generar ambos entornos utilizando el mismo modelo conceptual, definiendo un modelo de navegación para cada uno de ellos y cambiando la forma de generar el código fuente, por lo resulta de importancia que está herramienta permita seleccionar distintos templates a la hora de generar el código fuente justamente para poder enfocarlo a distintos entornos.

Como trabajo futuro se continuará con la programación de la herramienta de transformación y también se prevé agregar ciertas características de automatización para integrar la herramienta con ciertos entornos de forma que: 1. La base de datos sea automáticamente creada en un servidor SQL. 2. La aplicación se compile y esté disponible en forma automática en un servidor web sin que el usuario tenga que

compilar y hacer el deploy de la misma. Además se planea incorporar al modelo la posibilidad de trabajar con distintos roles de usuario permitiendo configurar los componentes a visualizar por cada uno de dichos roles.

Referencias

1. Backus J. W.: The Syntax and Semantics of the Proposed International Algebraic Language of the Zurich ACM-GAMM Conference Proceedings of the International Conference on Information Processing, UNESCO, pp.125-132. (1959)
2. Favre Liliana: Foundations of Component-Based Development and MDA, Proceedings of the 2005 Information Resources Management Association International Conference (2005), http://www.irma-international.org/viewtitle/32554/
3. Garzotto, D. Schwabe and P. Paolini: HDM - A Model Based Approach to Hypermedia Application Design. ACM Transactions on information Systems, pp. 1-26. (1993)
4. Jézéquel Jean-Marc, Defour Olivier and Plouzeau Noël: An MDA approach to tame component based software development. Post Proceedings of Formal Methods for Components and Objects (2004) http://hal.archives-ouvertes.fr/docs/00/79/50/36/PDF/Jezequel04d.pdf
5. Hain Mustapha, Belangour Abdessamad, Marzak Abdelaziz: Towards A Multiviews Component Based Model Driven Approach. International Journal of Engineering Science and Technology (2011), http://www.ijest.info/docs/IJEST11-03-01-159.pdf
6. Kleppe A., Warmer J., Bast W.: MDA explained: the model driven architecture: practice and promise. Addison-Wesley Professional (2003)
7. MIT: APP Inventor, http://appinventor.mit.edu/
8. Moreno Nathalie y Vallecillo Antonio: Towards Interoperable Web Engineering Methods. Journal of the American Society for Information Science and Technology (2008), http://www.lcc.uma.es/~av/Publicaciones/08/JASIST08.pdf
9. OMG: CORBA Version 3.3 (2012), http://www.omg.org/spec/CORBA/3.3/
10. OMG: MDA Guide Version 1.0.1 (2003), http://www.omg.org/cgi-bin/doc?omg/03-06-01
11. OMG: Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. Versión 1.1 (2011), http://www.omg.org/spec/QVT/1.1/PDF/
12. OMG: MOF 2 XMI Mapping. Version 2.4.1, http://www.omg.org/spec/XMI/
13. Schwabe D. y Rossi G. : An object oriented approach to Web-based applications design. Theor. Pract. Object Syst. Volume 4, Issue 4, pp 207-225. (1998)
14. Teiniker Egon, Mitterdorfer Stefan, Johnson Leif Morgan, Kreiner Christian, Kovács Zsolt: A Model Driven Component Framework for Mobile Computing. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.125.8612
15. W3C: Mobile Web Best Practices 1.0, http://www.w3.org/TR/mobile-bp/
16. W3C: Mobile Web Application Best Practices, http://www.w3.org/TR/mwabp/
17. Yang Zhihui: A Domain-Specific Modeling Approach For Component-Based Software Development. Ball State University Muncie Indiana (2009) http://cardinalscholar.bsu.edu/bitstream/123456789/193416/1/Zyang_2009-3_BODY.pdf