REVISTA COLOMBIANA DE COMPUTACIÓN
Volumen 16, número 2
Págs. 162 - 177

La Interfaz de Usuario como Punto de Partida para la Creación Automática de Aplicaciones Móviles – un Enfoque Basado en MDD

Pablo M. Vera[*], Claudia Pons[**], Carina S. González[***], Rocío A. Rodríguez[*]

Fecha de recibido: 01/06/2015 Fecha de aprobación: 05/07/2015

Resumen

Las aplicaciones móviles requieren una interfaz reducida y estandarizada para lograr que el usuario tenga una buena experiencia de uso. Particularmente las aplicaciones que administran datos utilizan controles de interfaz generales que permiten realizar todas las operaciones necesarias con dichos datos (listados, menús, pantallas de edición y de búsqueda). Tomando como base esos controles de interfaz de usuario se diseña una metodología de modelado basada en MDD (Model Driven Development), mediante la cual es posible generar automáticamente aplicaciones web móviles con el solo hecho de configurar los datos a visualizar en la interfaz y definir cómo será la navegación dentro del sistema. La configuración de los componentes está basada en un primer modelo que representa los datos que administrará la aplicación. Esta metodología está basada en UML (Unified Modelling Language) y se define mediante una extensión conservativa de dicho lenguaje especificada en un profile de UML y una serie de restricciones. Para soportar la metodología de modelado se ha desarrollado una herramienta que permite modelar y realizar las transformaciones entre modelos que son necesarios para poder generar el código fuente 100% funcional de una aplicación.

Palabras clave: *MDD, UML, Componentes, Móvil, Interfaz de usuario.*

---

[*]Universidad Nacional de La Matanza, GIDFIS (Grupo de Investigación, Desarrollo y Formación en Innovación de Software), Departamento de Ingeniería, Florencio Varela 1903, San Justo, Buenos Aires, Argentina. pvera@ing.unlam.edu.ar; rrodriguez@ing.unlam.edu.ar

[**]Universidad Nacional de La Plata, LIFIA (Laboratorio de Investigación y Formación en Informática Avanzada), Facultad de Informática, calle 50 y 115, Buenos Aires, Argentina. cpons@info.unlp.edu.ar

[***]Universidad La Laguna, ITED (Interacción, Tecnologías y Educación) Departamento de Ingeniería Informática y de Sistemas. Avda Astrofísico Fco. Sánchez s/n., La Laguna, 38204, Tenerife, España. cjgonza@ull.es

Abstract

Mobile applications requires a reduced and standardized interface to ensure that the user has a good user experience. Particularly applications that manage data uses general interface controls that allow performing all necessary operations with the data (lists, menus, screens for editing and search). Based on these controls a MDD (Model Driven Development) modelling methodology is designed, through which it is possible to automatically generate mobile web applications with the mere fact of configuring the data to be displayed in the interface and designing how will the system navigation be performed. The component configuration is based on a first model that represents the data that will be manage the application. This methodology is based on UML (Unified Modelling Language) and is defined by a conservative extension of the language specified in a UML profile and a number of restrictions. To support the methodology a tool was developed which allows modelling and performing the transformations between models that are needed to generate 100% functional source code of an application.

Keywords: *MDD, UML, Components, Móbile, User interface*.

1. Introducción

Al diseñar una aplicación es importante definir exactamente lo que se desea que el usuario vea en cada pantalla y cómo será el flujo de navegación entre las distintas pantallas del sistema. Esta tarea tan importante, es muchas veces relegada a los desarrolladores que a su criterio incorporan la información que les parece relevante. Sin embargo este es un proceso que debe ser no solo bien diseñado desde el punto de vista de la usabilidad sino que además debe ser consensuado con el usuario final para mostrar la información que realmente le sea relevante y brindarle todas las opciones y características para hacer su tarea más fácil.

Situándose en el área de las aplicaciones móviles, en especial aquellas aplicaciones que administran datos, las opciones se reducen ya que el tamaño de pantalla obliga a que las interfaces sean necesariamente simples y directas. Diversas fuentes académicas proveen principios importantes [1] [2] [3] [4] los cuales permiten establecer pautas para diseñar una interfaz móvil:

- No tener scroll horizontal.
- Mantener una barra de navegación mínima en la parte superior de la pantalla.
- Mantener una coherencia en los mecanismos de acceso y navegación entre las distintas pantallas.

Generalmente las aplicaciones que administran datos tienen un diseño estándar, con pantallas de listados, pantallas de edición, búsquedas, etc. Esto nos permite definir una serie de pantallas estándar que se utilizan en este tipo de aplicaciones:

- Listados de Información
- Pantallas de Búsqueda
- Pantallas de Edición
- Menús

Con estas pantallas es posible diseñar un gran conjunto de aplicaciones pero hay que buscar una forma para poder definir qué datos se desea visualizar en cada una de estas. Por ejemplo:

- En un listado se deben definir los datos a mostrar, cómo están ordenados, qué sucede al hacer click sobre un dato.
- En el menú se deben definir las distintas opciones y a qué pantalla conducirán cada una de ellas.
- En una búsqueda se deben definir cuáles son los filtros que el usuario podrá completar.
- En una pantalla de edición es posible que no todos los datos deban ser completados y por lo tanto eso debe también diseñarse.

Ahora bien, esa configuración de qué datos mostrar, va a requerir definir primero el modelo de datos la aplicación. El enfoque más utilizado para realizar este modelo es el diagrama de clases utilizados en muchas metodologías de modelado como por ejemplo OOHDM (Object Oriented Hipermedia Design Method) [5] y UWE (UML Based Web Engineering) [6]. Otras metodologías utilizan diagramas de entidad relación como WebML (Web Modeling Language) [7] y RMM (Relationship Management Methodology) [8]. Una vez definidas las clases del modelo de datos es posible detallar por ejemplo en un listado que datos mostrar refiriéndose a la clase y propiedad del modelo de datos deseados.

Teniendo en cuenta entonces que se dispone de una serie de pantallas donde no es necesario definir la ubicación de cada control a nivel interfaz ya que todas siguen un estándar y un diseño común, es posible diseñar una nueva metodología de modelado donde cada pantallas sea representada por un componente y cada componente pueda configurarse mediante un modelo de datos previamente definido.

2. Metodología de Modelado Basada en Componentes Configurables

En este trabajo se describe una metodología basada en el enfoque MDD (Model Driven Development) [9] en que partiendo de modelos se realizan distintas transformaciones para generar modelos más detallados hasta llegar a generar en forma automática el código fuente de una aplicación o al menos parte de dicho código. Podemos decir que "los modelos son creados no solo para capturar requisitos, sino también diseños e implementaciones, y no son solo meros artefactos utilizados en la documentación, sino documentos vivos que son transformados en implementaciones de sistemas" [10].

Esta metodología en particular tiene por objetivo la generación del código fuente completo de una aplicación. Es decir que solo con la información de los modelos sea posible generar código fuente 100% funcional, para lo cual será imprescindible contar con un modelo donde se disponga de toda la información necesaria.

La metodología se crea tomando como premisa la necesidad de generación automática del código fuente completo de una aplicación y además con el objetivo de simplificar el proceso de modelado. Esta metodología utiliza componentes configurables predefinidos de interfaz de usuario para establecer el comportamiento del sistema donde cada componente se corresponde con una pantalla a visualizar por el usuario, y la hemos denominado CBMDD (Component Based Model Driven Development).

La utilización de componentes en el proceso de desarrollo de software es una técnica muy bien establecida en la industria de software [11]. Los componentes son piezas de software predefinidas con un propósito puntual. "El desarrollo de software basado en componentes, permite que la construcción del software se realice mediante el ensamblado de bloques prefabricados configurables y de evolución independiente" [12].

Los componentes son excelentes para la reutilización y son confiables ya que una vez que son testeados y su comportamiento fue verificado, se pueden utilizar en un gran número de aplicaciones sin modificación. "Los beneficios de los componentes de software incluyen la reutilización y la interoperabilidad, entre otros" [13].

La mayoría de los frameworks de programación modernos incluyen componentes predefinidos para facilitar el proceso de desarrollo. Por ejemplo componente de autenticación en el sistema o componentes de

interfaz de usuario como grillas, carruseles, etc. Análogamente se propone utilizar componentes configurables para facilitar la generación de código fuente en MDD.

El uso de componentes en MDD agrega las ventajas intrínsecas del desarrollo basado en componentes al proceso de desarrollo dirigido por modelos, entre las principales ventajas se pueden mencionar:

- Facilita el desarrollo, prueba e implementación de las transformaciones; ya que una vez desarrollada la transformación se reutiliza en todas las aplicaciones posteriores.

- Facilita el modelado al tener bloques prefabricados que pueden configurarse para adaptarse a distintas aplicaciones.

Pero antes de poder configurar los componentes debe definirse el modelo de datos, a partir del cual se podrá mediante una transformación generar en forma automática un diagrama con componentes estándar llamado modelo de interfaz de usuario. Los pasos de la metodología pueden verse en la Fig. 1.

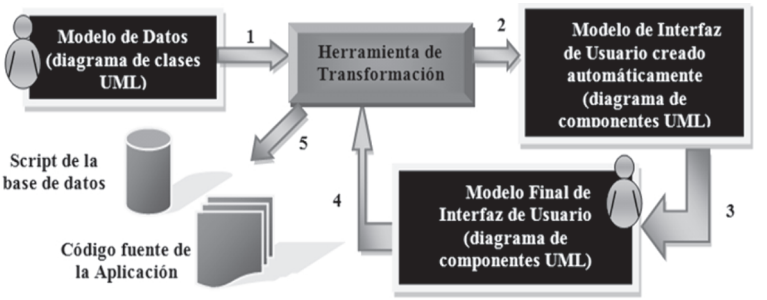

Fig. 1. Pasos de la metodología de modelado.

Los pasos a realizar son los siguientes:

1. El diseñador realiza el modelo de datos del sistema.

2. Tomando como base el modelo de datos, la herramienta de transformación genera de forma automática una primera versión del modelo de interfaz.

3. El diseñador realiza los ajustes al modelo de interfaz, cambiando la configuración y/o agregando nuevos componentes.

4. Tomando los dos modelos terminados, se realiza la segunda y última transformación que genera el script de la base de datos y el código fuente de la aplicación.

Ambos diagramas están basados en diagramas tradicionales de UML pero extendidos mediante estereotipos y valores etiquetados. Todas las extensiones realizadas se definen mediante el perfil de UML de la Fig. 2. Todos los nuevos estereotipos definidos heredan de alguna metaclase de UML generando de esta forma una extensión conservativa. "Conservativa significa que los elementos de modelo del metamodelo de UML no se modifican… todos los nuevos elementos se relacionan por herencia hacia al menos un elemento del modelo del metamodelo de UML" [14].

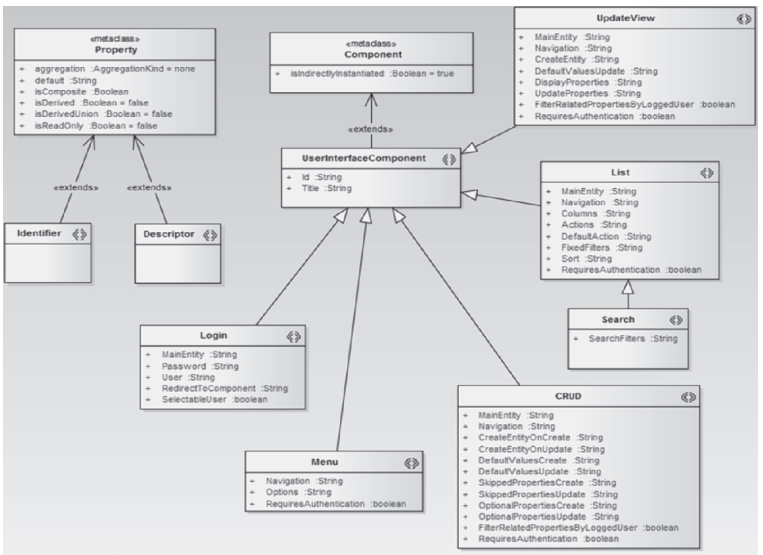

Fig. 2. Perfil UML de la metodología de modelado.

El perfil de UML tiene dos extensiones una para el modelo de datos y otra para el modelo de interfaz de usuario. La extensión para el modelo de datos agrega dos estereotipos que se aplican a las propiedades de las clases por ello extienden la metaclase property. Los dos estereotipos son Identifier y Descriptor. Para el modelo de interfaz de usuario se parte de la metaclase Component y se la extiende con un estereotipo abstracto llamado UserInterfaceComponent que contiene los valores etiquetados comunes de cada uno de los estereotipos que representan cada uno de los componentes. De este estereotipo abstracto heredan el resto de los estereotipos donde para cada uno se declaran los valores etiquetados propios necesarios para poder configurarlos. En el caso particular del estereotipo Search este hereda todos los valores etiquetados del estereotipo List y agrega un único valor etiquetado particular.

Para completar el perfil se agregan algunas restricciones expresadas en lenguaje natural:

1. El estereotipo Identifier solo puede aplicarse a propiedades del tipo Integer.

2. El estereotipo Descriptor solo puede aplicarse a propiedades del tipo String.

3. Una clase puede tener una única propiedad con el estereotipo Identifier y la misma es obligatoria.

4. Una clase puede tener una única propiedad con el estereotipo Descriptor y la misma es obligatoria.

5. El valor etiquetado Id de un componente no puede repetirse con los valores etiquetados Id del resto de los componentes.

6. El valor etiquetado Title de un componente no puede repetirse con los valores etiquetados Title del resto de los componentes.

7. Solo puede existir un componente con el estereotipo Login.

2.1 Modelo de Datos

El primer paso es realizar el modelo de datos de la aplicación, para ello se utiliza el diagrama de clases de UML donde cada clase representa una tabla de la base de datos. En este diagrama se define información adicional que luego facilitará la creación de la base de datos y la visualización de información en el sistema. Esta información adicional se expresa en el diagrama de clases mediante estereotipos que permiten determinar la clave primera de una clase utilizando el estereotipo Identifier y una descripción textual de dicha clase mediante el estereotipo Descriptor. Adicionalmente se utiliza el estereotipo enumeration ya presente en UML para definir, por ejemplo, distintos estados de un modelo.

2.2 Modelo de Interfaz de Usuario

En este modelo se diseñarán las distintas pantallas que conformaran el sistema, determinando además, cuál es el flujo de navegación entre dichas pantallas. Para modelar la interfaz de usuario se utilizará el diagrama de componentes de UML, donde cada componente se corresponderá con una pantalla del sistema.

Para identificar el tipo de pantalla que se desea modelar se crearon estereotipos que se aplican a los componentes de UML para tipificarlos.

Los estereotipos disponibles son: Login, List, Search, Menu, CRUD y UpdateView.

Cada tipo de componente tiene definida una serie de valores etiquetados que permite configurar su comportamiento para adaptarlo a la aplicación que se desea modelar. Algunos de estos valores etiquetados son comunes para la mayoría de los componentes y otros son particulares para cada uno de ellos. El detalle de configuración de cada componente y de sus valores etiquetados puede verse en [15].

Existen algunas opciones de configuración que son comunes como ser un título general del componente, que se utilizará para mostrar al usuario en la pantalla que se encuentra y la barra de navegación que está presente en todos los componentes que se configura mediante una serie de links que llevan a otros componentes. Además cada componente incluye opciones de configuración propia que permiten definir cómo será la interfaz de usuario. A continuación se enumeran dichas opciones de configuración y se muestran pantallas generadas automáticamente por la herramienta de modelado de CBMDD.

- Login: Este componente permite la autenticación de un usuario en el sistema. Además de permitir configurar contra qué tabla se realizará la autenticación incluye la posibilidad de permitir que el usuario en lugar de ser ingresado, sea seleccionado, esto es útil en los dispositivos móviles ya que el ingreso de texto es una tarea dificultosa. La Fig. 3 muestra dos pantallas de login, una en la cual el usuario debe ser ingresado y la otra donde directamente se selecciona.

Fig. 3. Pantallas generadas automáticamente a partir de componentes del tipo Login.

- List: Este componente muestra listados de información. Su configuración permite:
  - Definir qué datos se van a mostrar.
  - Definir datos adicionales que se pueden mostrar en una segunda fila de información correspondiente a cada ítem.

- Establecer filtros por defecto que se aplican al listado visualizado.
- Determinar el orden del listado.

La Fig. 4 muestra dos ejemplos de pantallas generadas a partir de componentes del tipo List. La imagen de la izquierda muestra un listado donde la información se muestra en varias líneas, una línea principal con un título y una línea adicional mostrando información complementaria, este caso con la categoría y fecha de vencimiento. En el lado derecho de la imagen se ve un listado con un solo dato mostrando la descripción de las categorías, pero en la barra de navegación además de la opción para volver al menú principal hay un botón para ir a la pantalla que permite agregar una nueva categoría.

Fig. 4. Pantallas generadas automáticamente a partir de componentes del tipo List.

- Search: es un componente que permite mostrar un listado de información pero el mismo puede ser filtrado en tiempo de ejecución por el usuario. Incluye la misma configuración del componente List pero capacidad de definir sobre que campos se permitirá realizar la búsqueda y el tipo de control asociado al mismo. Los tipos de filtros disponibles a aplicar son:
  - Texto Libre
  - Selección Simple
  - Selección Múltiple
  - Rango de Fechas
  - Verdadero o Falso.

La Fig. 5 muestra un ejemplo de los filtros de búsqueda que se generan en la interfaz de la aplicación.

Fig. 5. Pantalla de búsqueda generada automáticamente a partir de un componente Search.

- Menu: este componente permite mostrar al usuario una pantalla con un menú. Su configuración es simple ya que solo se deben definir las opciones de dicho menú mediante una serie de links que apuntan a otros componentes tal como puede verse en la Fig. 6.

Fig. 6. Pantalla generada automáticamente a partir de un componente del tipo Menú.

- CRUD: es un componente para crear, modificar, visualizar y eliminar datos. Por defecto este componente solicita al usuario el ingreso de todas las propiedades de la clase que se está editando, pero incorpora una potente configuración que permite:
  - Definir valores por defecto al crear o actualizar un objeto.
  - Evitar llenar completar determinadas propiedades de una clase al crear o actualizar un objeto.
  - Crear registros adicionales de otras tablas al crear o actualizar un objeto, lo que es muy útil para crear tablas de log.
  - Definir propiedades opcionales.

La Fig. 7 muestra una pantalla de edición donde para cada propiedad se generan automáticamente los controles adecuados para el usuario pueda ingresar los datos. Puede verse que para clases relacionadas se genera un control de selección con la descripción de la otra entidad como es el caso de la categoría.

Fig. 7. Pantalla de edición generada automáticamente a partir de un componente del tipo CRUD.

- UpdateView: este componente permite realizar operaciones de actualización sobre objetos incorporando características especiales que lo diferencian del CRUD. En muchos sistemas es posible que un único objeto deba ser actualizado parcialmente donde solo algunas de sus propiedades se deban completar y otras sean mostradas en modo de solo lectura. Este control es muy útil para sistemas donde un objeto va pasando por diferentes estados y en cada estado se registran distintos valores. Para actualizar cada uno de esos estados podría configurarse un componente UpdateView diferente. La Fig. 8 muestra un ejemplo de una pantalla generada por un componente UpdateView donde se configuraron 3 propiedades de solo lectura (título, categoría y fecha de vencimiento) y solo se permite editar al usuario la descripción. En este caso, además, al actualizar los datos el estado de la tarea cambia automáticamente a Completa.

Fig. 8. Pantalla generada automáticamente a partir de un componente del tipo UpdateView.

3. Herramienta de Modelado

La metodología de modelado fue desarrollada basada en UML utilizando una extensión conservativa mediante estereotipos y valores etiquetados, esto hace que sea posible crear los modelos en cualquier herramienta basada en UML ya que los mecanismos de extensión están presentes en todas ellas.

Sin embargo la configuración de cada valor etiquetado es particular y requiere recordar la sintaxis para realizar una configuración correcta. Este es un primer motivo por el cual se decidió desarrollar una herramienta que permita facilitar dicha configuración, el segundo motivo es que dicha herramienta dará soporte además a las trasformaciones entre modelos.

La herramienta es una aplicación Web desarrollada en C# bajo el framework Microsoft ASP.NET V4 utilizando Web Forms [16]. La misma fue desarrollada con los siguientes objetivos:

1. Facilitar el modelado evitando que el diseñador deba recordar la configuración particular de cada componente.

2. Permitir la interoperabilidad con otras aplicaciones de modelado mediante XMI.

3. Validar la correcta configuración de los componentes en el caso de que se realice con una herramienta externa.

4. Realizar las transformaciones automáticas previstas en la metodología.

Además la herramienta fue desarrollada para ser escalable permitiendo:

- Agregar fácilmente nuevos componentes.

- Permitir reutilizar controles para configurar los componentes.

- Poner a disposición clases para generar fácilmente código fuente en cualquier lenguaje.

La Fig. 9 muestra la pantalla principal de opciones de un proyecto donde claramente pueden verse las opciones para administrar tanto el modelo de datos (Clases) como el de interfaz de usuario (Componentes) y además poder realizar las transformaciones correspondientes.

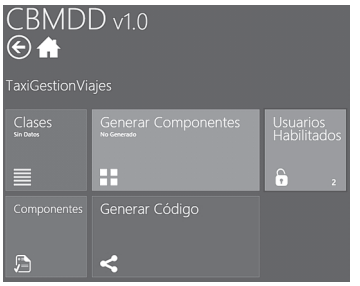

Fig.9. Menú de acciones de un proyecto de modelado.

4. Resultados

Se desprende de las secciones anteriores que mediante la herramienta construida es posible realizar los modelos y aplicar las transformaciones. Generándose los scripts de la base de datos y el código fuente de la aplicación. Por otra parte al comenzar un proyecto la herramienta muestra los estados de los diagramas (por ejemplo: diagrama de clases – sin datos; Generar Componentes – no generado; etc.). Estos estados cambian a medida que se trabaja con la herramienta. También permite evitar por ejemplo que el usuario cambie el diagrama de clases y no vuelva a generar el de componentes. Permitiendo que exista una trazabilidad entre los modelos y transformaciones aplicadas (ver Fig. 10).

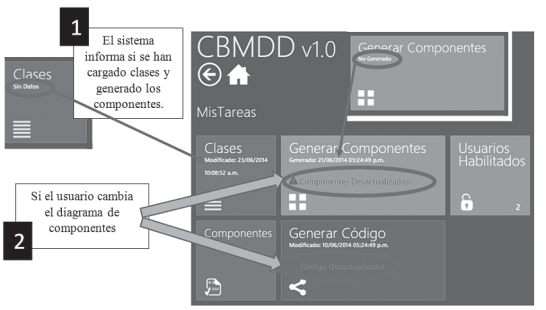

Fig.10. Menú de acciones de un proyecto de modelado.

Se han realizado diversas aplicaciones con distintos grados de complejidad pudiendo generarse el código fuente funcional de ellas a través de la herramienta, también se ha trabajado en la interoperabilidad permitiendo que pueda exportarse el XMI del modelo y luego importarse en otra herramienta de modelado. Para esto se creó un parser de XMI 2.2 que brinda interoperabilidad con el EA (Enterprise Architecture) permitiendo importar en la herramienta los modelos previamente construidos EA o exportarlos al EA.

Las opciones para importar y exportar modelos mediante XMI se agregaron tanto para el modelo de datos como para el modelo de componentes. En el caso del proceso de importación también se realiza una validación para comprobar que el modelo que se está importando esté correctamente configurado. En la Fig. 11 se muestran las opciones en pantalla que permiten poder importar o exportar el modelo de datos mediante XML.

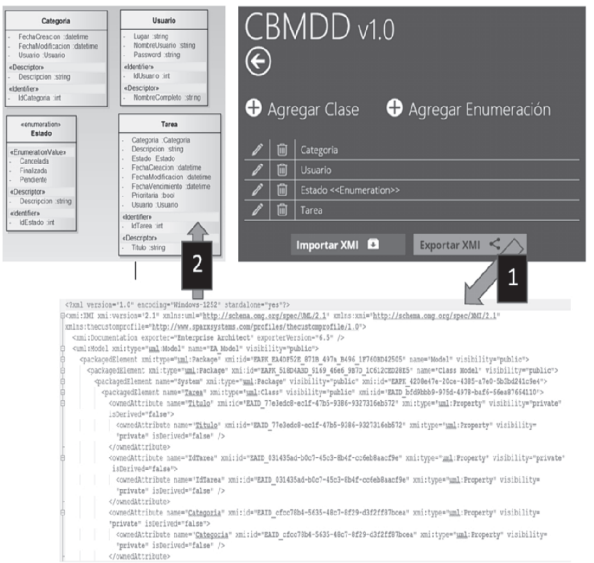

Fig. 11. Exportación de modelo de datos al EA.

5. Conclusiones

MDD permite que el esfuerzo esté puesto en las etapas iniciales de elicitación de requerimientos y modelado; disminuyendo el tiempo y esfuerzo del desarrollo en sí mismo. El problema en muchos caso es cuán alto es ese esfuerzo inicial comparativamente con el que se debería realizar para desarrollar sin que los modelos sean la parte central de trabajo. En muchos casos las metodologías basadas en MDD proponen una gran cantidad de modelos con una engorrosa relación entre ellos, en donde el esfuerzo de realizarlos para muchos desarrolladores experimentados es mayor que el que ellos invertirían en desarrollar una aplicación de cero. Este es el gran valor de la presente metodología tan solo dos modelos que se pueden realizar en cualquier herramienta de modelado con una extensión conservativa de UML que puede ser facilitada desde la herramienta creada. La particularidad de la metodología por otra parte es la reutilización de componentes configurables y la aplicabilidad de transformaciones automáticas

generadas desde una herramienta, disminuyendo de este modo el esfuerzo del usuario.

Por otra parte es importante considerar a los usuarios móviles y que las aplicaciones sean prácticas, para un usuario en un contexto en movimiento. Las interfaces deben proponer una navegación fácil y clara, con una barra de navegación reducida posicionada en la parte superior de la pantalla. Ponerse en el lugar del usuario final y considerar los componentes necesarios para las aplicaciones centradas en los datos ha sido el punto de partida de esta propuesta, esto es, una clave importante ya que si el diseño de las pantallas no son consistentes, la navegación no es simple y ágil y no se ofrecen mecanismos de búsqueda y filtros; este tipo de aplicaciones no serían utilizadas y por ende carecería de sentido analizar el tiempo de desarrollo de las mismas.

Actualmente el código final es desarrollado en HTML 5 con JQuery Mobile, pero pueden generarse templates que permitan al usuario seleccionar el código de destino, esto es una futura línea de acción.

Referencias

[1] W3C. Mobile Web Best Practices 1.0. [Online] 2009. http://www.w3.org/TR/mobile-bp/.

[2] W3C. Mobile Web Application Best Practices. [Online] 2008. http://www.w3.org/TR/mwabp/.

[3] Krug, S. No me hagas pensar: una aproximación a la usabilidad en la web. s.l. : Pearson Educación, SA., 2001.

[4] Budiu, R., Nielsen, J. Usability of iPad apps and websites. 2011.

[5] Schwabe D, Rossi G. An Object Oriented Approach to Web-Based Application Design. 1998, pp. 207-225.

[6] N., KOCH. Software Engineering for Adaptive Hypermedia Applications. s.l. : PhD. Thesis, Reihe, 2001.

[7] Ceri S., Fraternali P., Bongio. Web Modeling Language (WebML): a modeling language for designing Web sites. 2000. pp. 137-157.

[8] Isakowitz, E. Stohr A., Balasubramanian P. RMM: a methodology for structured hypermedia design. 1995. s.l. : ACM, pp. 34-44.

[9] Brambilla, M. C. Model-driven software engineering in practice. s.l. : Synthesis Lectures on Software Engineering, 2012. Vol. 1.

[10] Agrawal, A., Levendovszky, T., Sprinkle, J., Shi, F., & Karsai, G. Generative programming via graph transformations in the modeldriven architecture. USA : Workshop on Generative Techniques in the Context of Model Driven Architecture, OOPSLA, 2002.

[11] Heineman, George T and Councill, William T. Component-Based Software Engineering: Putting the Pieces Together. s.l. : Addison-Wesley, 2001.

[12] Keller, Rudolf K and Reinhard, Schauer. Design components: toward software composition at the design level. s.l. : IEEE Computer Society, 1998.

[13] Adler, Richard M. Emerging standards for component software. 1995. pp. 68-77. Vol. 28.

[14] Rossi, G., et al. Web Engineering: Modelling and Implementing Web Applications. 2008.

[15] Vera P., Pons C. Gonzalez C, Giulianelli D., Rodriguez R. MDA based Hypermedia Modeling Methodology using reusable components. s.l. : XVIII Congreso Argentino de Ciencias de la Computación, 2012.

[16] Microsoft. ASP.NET Web Forms. [Online] [Cited: 01 05, 2015.] http://www.asp.net/web-forms.