Desarrollo Dirigido por Modelos Basado en Componentes de Interfaz de Usuario

Tesis presentada para obtener el grado de

Doctor en Ciencias Informáticas

Tesista: Pablo Martín Vera

Directora: Dra. Claudia Pons

Co-Directora: Dra. Carina González González

Facultad de Informática

UNIVERSIDAD NACIONAL DE LA PLATA

Junio del 2015

RESUMEN

Esta tesis presenta una metodología de modelado para aplicaciones web móviles utilizando técnicas de desarrollo dirigido por modelos (MDD). Mediante la creación de sólo dos diagramas, un diagrama de datos y un diagrama de interfaz de usuario (que además incluye la navegación) es posible definir el comportamiento completo de una aplicación.

Por estar esta metodología basada en MDD incorpora dos transformaciones; la primera desde el modelo de datos a una versión inicial del modelo de interfaz de usuario, lo que reduce considerablemente el esfuerzo de modelado, ya que luego ese segundo modelo solo deberá ser adaptado a las necesidades particulares. La segunda transformación toma los modelos realizados y genera el código fuente completo, 100% funcional de una aplicación web móvil, además del script de la base de datos correspondiente. Ambos modelos están basados en una extensión conservativa de UML. El modelo de datos está basado en el diagrama de clases y el modelo de interfaz de usuario utiliza el diagrama de componentes de UML. Para poder especificar el comportamiento de la interfaz de usuario se definen una serie de componentes que a su vez pueden ser configurados con información tomada del modelo de datos. La configuración se basa en valores etiquetados propios para cada tipo de componente. Para facilitar el proceso de construcción de los modelos y su configuración se ha desarrollado una herramienta de soporte, que permite no solo modelar, sino también realizar las transformaciones establecidas en la metodología, obteniendo como resultado final una aplicación funcional sin escribir una sola línea de código.

Además esta tesis establece las ventajas de utilizar componentes configurables en el desarrollo dirigido por modelos, haciendo que el esfuerzo de programación se realice una única vez al establecer las transformaciones y que luego pueda ser aplicado a una amplia gama de aplicaciones de distintos dominios.

DEDICATORIA

A mi familia, amigos y colegas, *quienes estuvieron alentándome a lo largo de este camino.*

A Claudia Pons, *quién es un referente en esta área y ha sido una gran directora de tesis, siempre con una actitud amable y positiva.*

A Carina González González, *quién ha logrado que no se noten las distancias físicas, estando siempre dispuesta a contestar dudas, favorecer el debate y realizar recomendaciones importantes****.***

AGRADECIMIENTOS

Esta tesis es el resultado de un largo camino transitado, en el cual recibí el apoyo en cuanto a lo académico por parte las siguientes instituciones:

- Universidad Nacional de La Plata (UNLP), agradeciéndole a sus autoridades, docentes y personal administrativo; por recibirme cálidamente a lo largo de la cursada.
- Universidad Nacional de La Matanza (UNLaM), agradeciéndoles a sus directivos por estimular a sus docentes investigadores a realizar estudios de postgrado.
- Universidad Tecnológica Nacional (UTN), en donde no realizo actividades de investigación y no obstante siempre alentaron mi participación en eventos académicos, aun cuando estos suponían alguna ausencia en mis actividades en el aula.
- A la Universidad Abierta Interamericana (UAI) en donde comencé hace unos años a dar clases de postgrado y luego me incorporé como co-director en un proyecto relacionado con tecnológicas móviles en el CAETI (Centro de Altos Estudios en Tecnología Informática) y las autoridades tanto de la UAI como del CAETI siempre han estado atentos a mis avances y estimulando mis progresos.

Así también debo agradecer desde lo humano a mis compañeros de las universidad mencionadas previamente (docentes, becarios y pasantes) quienes comparten mucho de su tiempo conmigo y sé que ahora comparten además mi alegría y satisfacción de haber culminado la presente tesis doctoral.

Sin lugar a dudas toda tesis es producto de un gran esfuerzo realizado no sólo por el tesista sino por los directores de la misma. Por ello debo agradecerles a Claudia Pons y Carina González Gonzales, por su profesionalismo y calidad humana.

Finalmente agradezco a mis alumnos de grado y postgrado quienes siempre son una fuente de motivación. Y a todos aquellos que de alguna forma me han acompañado en este largo camino.

INDICE

Índice de Figuras

Índice de Tablas

Capítulo 1 – Introducción

1.1 Definición del Problema

El modelado de aplicaciones es un área subestimada por la industria de software. Especialmente en pequeñas y medianas empresas donde el modelado es considerado, en numerosas oportunidades, una pérdida de tiempo. En otros casos, los modelos solo se utilizan en etapas tempranas del desarrollo para la toma de requerimientos o como documentación inicial que luego no se actualiza con los cambios realizados durante el desarrollo de la solución, por lo que rápidamente quedan obsoletos. Por esa razón surge la idea de dar más importancia a los modelos. Los modelos pueden ser utilizados para generar de forma automática una aplicación o por lo menos parte de ella. "La automatización del proceso de desarrollo de software consiste en comenzar desde un alto nivel de representación de las características deseadas del software y derivar una aplicación ejecutable a partir de ella, posiblemente mediante un conjunto de pasos intermedios que permitan algún grado de interacción del usuario con el proceso de generación" (Brambilla, 2012).

El desarrollo de software mediante la construcción de modelos que permitan luego la generación automática de código fuente a partir de los mismos, es una tendencia iniciada hace varios años atrás. Puede encontrarse con diversos nombres: MDD (Model Driven Development), MDA (Model Driven Architecture), MDSE (Model Driven Software Engineering), MDE (Model Driven Engineering), entre otras. Sin embargo estas siglas tienen una categorización y relación entre ellas que son detalladas en la sección 2.1.

Las técnicas mencionadas tienen algo en común, todas utilizan modelos y transformaciones.

Un modelo es una representación abstracta de la realidad, en este caso los modelos que utilizaremos son modelos software y por lo tanto podrán representar partes o componentes de un sistema. Un modelo puede tener una representación gráfica y también una descripción textual, lo importante es que siga ciertas reglas para su conformación.

Una transformación es un proceso que toma como entrada un modelo y genera como resultado otro modelo o código fuente. Por ejemplo el enfoque MDA de la OMG (Kleppe, Warmer, & Wim, 2003) (OMG, MDA Guide Version 1.0.1, 2003), utiliza diferentes tipos de modelos con diferentes niveles de abstracción. Partiendo de modelos independientes de la plataforma (PIM) hasta llegar a modelos específicos para cada plataforma (PSM). "El PIM permite una representación visual del modelo, utilizando un nivel alto de abstracción. Los detalles de los modelos ambientales pueden ser expresados y precisados claramente en UML sin utilizar ningún formalismo particular de la implementación… un PSM se desarrolla mediante un mapeo de un PIM a una plataforma computacional y a un lenguaje de programación específico" (Papajorgji, Beck, & Braga, 2004).

Para llegar desde el PIM al PSM se deben realizar transformaciones automáticas o semi-automáticas. El objetivo final de estas técnicas es automatizar el proceso de generación del código fuente permitiendo que los diseñadores se centren en los modelos, más que en proceso de codificación.

Sin embargo, el principal problema, es que la mayoría de las técnicas existentes son difíciles de utilizar y requieren de un arduo proceso detallando modelos y configurando transformaciones para lograr obtener código fuente utilizable; y aun así la mayoría de ellas solo permite generar parte del código fuente. Algunos autores comparten la visión de que las metodologías de modelado para generación de código fuente son demasiado complejas "Para realizar un programa útil partiendo de un modelo, el modelo debe ser tan complicado que la gente que no puede programar no lo puede comprender y los que pueden programar preferirán escribir el código antes que realizar los correspondientes modelos" (Steimann & Kühne, 2005).

Es por este motivo que surge la necesidad de crear una metodología de modelado fácil de utilizar, que no requiera de modelos complejos y que como resultado se pueda obtener el código fuente completo de una aplicación.

1.2 Hipótesis

La hipótesis de trabajo de esta tesis es la siguiente:

Es posible desarrollar una metodología de modelado que incluya toda la información necesaria para generar el código fuente completo de una aplicación realizando transformaciones automáticas, con pocos modelos, simples y fácilmente entendibles.

1.3 Solución Propuesta

Como solución se plantea el diseño de una metodología de modelado basada en componentes de interfaz de usuario configurables que incluye toda la información necesaria para generar el código fuente completo de una aplicación. Esta metodología se denomina CBMDD (Component Based Model Driven Development)

1.4 Alcance

La presente tesis propone una nueva forma de utilizar MDD mediante el uso de componentes configurables y define una metodología orientada principalmente a aplicaciones web móviles centradas en los datos. En la sección 3.3 se expone en detalle el motivo por el cual la metodología se enfoca en este tipo de aplicaciones.

1.5 Contribución Científica

Las principales contribuciones científicas de la siguiente tesis son:

- Utilización de componentes configurables para facilitar la generación de código fuente en MDD
- Diseño de una metodología para el modelado de aplicaciones web móviles centradas en los datos.

1.6 Enfoque Metodológico

La Figura 1 presenta la metodología empleada para la realización de la presente tesis doctoral. Para simplificar el esquema no se han representado líneas de retroalimentación en cada una de las tareas realizadas.

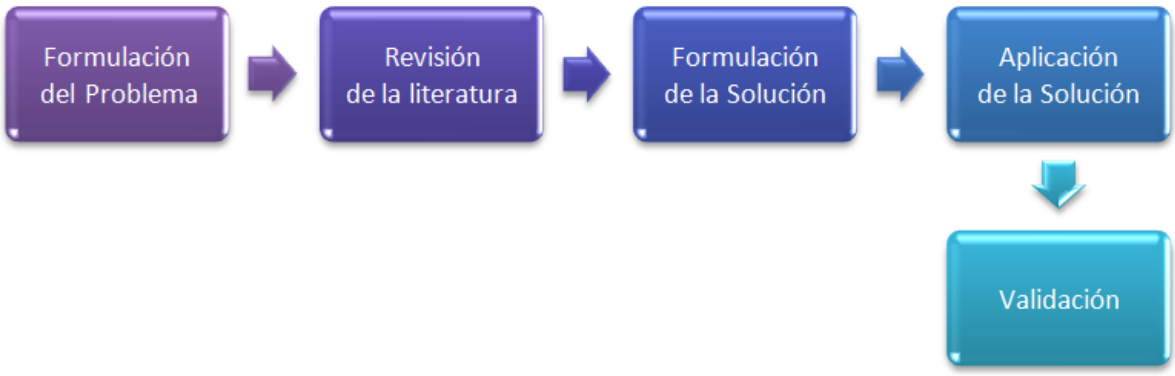

Figura 1. Metodología empleada para la realización de la tesis

1. Formulación del Problema: definición del problema a tratar que fue explicado en la sección 1.1
2. Revisión de la Literatura: Búsqueda, selección, recopilación y revisión de trabajos relacionados pertinentes.
3. Formulación de la solución: Para el diseño de la metodología de modelado se ha utilizado un proceso iterativo incremental. Se comenzó determinando para una aplicación sencilla como debería realizarse el modelado y que características eran necesarias hacer explícitas para tener la información suficiente para generar el código fuente. Una vez definida la primera versión de la metodología se intentó aplicarla a una aplicación más compleja haciendo necesario realizar agregados y ajustes a la metodología original. Este cíclico se realizó varias veces con aplicaciones cada vez más complejas lo que permitió ir refinando la metodología.
4. Aplicación de la Solución: Una vez definida completamente la metodología se realizó un prototipo de prueba de una herramienta de generación de código fuente lo que a su vez permitió realizar los ajustes finales al modelo.
5. Validación: Contrastación de los resultados obtenidos con la hipótesis de la tesis.

1.7 Estructura de la Tesis

Esta tesis está organizada en 6 capítulos. A continuación se describe brevemente el contenido de cada uno de ellos:

- Capítulo 1 - Introducción: En este capítulo se define el problema de estudio, estableciendo una hipótesis de trabajo y proponiendo una solución para llevar a cabo la misma. Se define el alcance de la propuesta, su contribución científica y el enfoque metodológico empleado.

- Capítulo 2 - Marco Teórico: En este capítulo se introduce al enfoque MDD y se presentan las principales metodologías de modelado existentes, finalmente se analizan trabajos relacionados en esta área.
- Capítulo 3 – Propuesta: Se comienza proponiendo el uso de componentes en MDD hasta llegar a establecer una metodología de modelado para aplicaciones web móviles. Se detallan los pasos, modelos y transformaciones de dicha metodología.
- Capítulo 4 – Implementación: Muestra el desarrollo de una Herramienta de soporte para la utilización de la metodología. La que incluye soporte para la construcción de los modelos, trazabilidad, transformaciones entre modelos y a código fuente. También se incluye un apartado sobre interoperabilidad con otras herramientas.
- Capítulo 5 – Validación: En este capítulo se valida la propuesta realizada, mediante distintos aspectos. Primero se comprueba la validez del lenguaje de modelado. Luego se comprueba mediante un ejemplo la efectividad de la propuesta, además de demostrar la aplicabilidad de la misma a distintos dominios y distintos esquemas de datos. Por último se enumeran una serie de características que resaltan la flexibilidad de la metodología planteada.
- Capítulo 6 - Conclusiones y Trabajos Futuros: Se presentan las conclusiones finales sobre el trabajo realizado. También en este capítulo se listan las publicaciones realizadas relacionadas con el tema de la presente tesis y se plantean trabajos futuros.

Finalmente se presentan los ACRONIMOS utilizados a lo largo de la tesis, las REFERENCIAS y dos ANEXOS cuyos contenidos se describen a continuación:

- Anexo A comparativa de XMI entre distintas herramientas: Resume el trabajo de comparación realizado con las principales herramientas de modelado UML al exportar los modelos con las extensiones requeridas por CBMDD al formato XMI.
- Anexo B Modelado de Aplicaciones: Muestra diversos modelos de ejemplo al aplicar la metodología a aplicaciones de distintos dominios.

Capítulo 2 - Marco Teórico

2.1 Desarrollo Dirigido por Modelos (MDD)

El desarrollo de software es un proceso que está formado por diferentes etapas: Análisis, Diseño, Desarrollo, Pruebas e Implementación. En la etapa de análisis se determina cual es el problema a resolver cuya solución se plantea en la etapa de Diseño. Es en el diseño, donde se decide cómo se va a resolver el problema, en que plataforma se va a desarrollar, que módulos va a contener, y también que pantallas va a contener esa aplicación y como serán esas pantallas. Una vez tomadas todas las decisiones de diseño se procede al desarrollo que no es ni más ni menos la programación de la solución en un lenguaje de programación específico. Una vez terminada la programación se realizan pruebas para corroborar que el sistema desarrollado no solo funcione correctamente sino que además cumpla con los objetivos establecidos. Por último la etapa final de todo proceso software es la implementación, que incluye la puesta en marcha en un ambiente productivo de la solución. Esta etapa es donde el software comienza a utilizarse realmente y debido a uso pueden surgir adaptaciones, cambios o mejoras que harán que el ciclo de desarrollo vuelva a comenzar con un nuevo análisis de los ajustes necesarios.

En el proceso de desarrollo, ya desde las primeras etapas, se comienzan a utilizar modelos para poder simplificar el problema, comprenderlo y diseñar una posible solución. Como se mencionó en el capítulo anterior, un modelo es una abstracción de la realidad que simplifica la misma haciéndola comprensible y manejable. Existen distintos tipos de modelos que pueden utilizarse en el desarrollo de software pero a partir del nacimiento de la programación orientada a objetos nace una notación universal estándar mediante el Lenguaje Unificado de Modelado (UML) (OMG, Unified Modeling Language - Resource Page, s.f.). UML es un lenguaje de modelado que incluye representaciones gráficas que permiten especificar las distintas partes de un sistema, incluyendo su estructura y su comportamiento.

Tanto en las etapas de análisis como de diseño se utilizan modelos que luego son tomados por los programadores para escribir el código fuente siguiendo las instrucciones de los mismos. Principalmente los modelos de diseño son los más utilizados por los programadores ya que definen desde cómo debe estar estructurada la aplicación, los datos que maneja, la comunicación entre módulos, hasta cómo será la interfaz de usuario, definiendo las distintas pantallas en las que el usuario final interactuará con el sistema. Análogamente a la arquitectura edilicia, podría decirse que los modelos son los planos del software que les indican a los programadores como exactamente deben construir la solución propuesta.

Para lograr modelos que cubran todos los aspectos del diseño de un sistema, el trabajo necesario en la etapa de diseño es muy grande y en muchos casos no se realiza completamente dejando decisiones para la etapa de codificación. Esta delegación de responsabilidades puede provocar software que no cumpla con lo solicitado con el cliente ya que los programadores generalmente no están en contacto con el usuario final. Además las decisiones tomadas directamente en la etapa de codificación no quedan documentadas en los modelos del software por lo que futuras modificaciones se hacen más complejas.

Muchas empresas no entienden la necesidad de realizar modelos detallados o no lo realizan simplemente por falta de tiempo y/o encarecimiento del proceso de desarrollo.

“La mayoría de los desarrolladores operan sentándose en su editor de texto favorito y escribiendo sus programas, intentando compilarlo, haciendo cambios, compilándolo, testeándolo y así hasta que el programa funcione. Algunas veces las distintas razones de las decisiones de diseño son capturadas en comentarios en el en código o en otros documentos. A menudo perdidos en la prosperidad. Esas fundamentaciones y decisiones de diseño son, sin embargo, críticas para el éxito de producto de programación de larga vida y alta calidad. Es por eso que el enfoque estándar *laissez faire* en la programación que muchos practicantes han aprendido, debe ser reemplazado por una metodología ingenieril más disciplinada”. (Hailpern B., 2006)

Para evitar estos problemas surge el desarrollo dirigido por modelos (MDD) donde lo que se busca es la reutilización de los modelos utilizados en las primeras etapas de análisis y diseño para poder generar automáticamente el código fuente de una aplicación.

MDD propone partir de modelos, que pueden tener distintos niveles de abstracción según la etapa del proceso que se esté modelando. Además MDD propone la utilización de transformaciones para que partiendo de un modelo ya terminado se pueda generar de forma automática parte de otro modelo necesario para especificar el sistema, lo que reduce también el trabajo de modelado. Esta evolución de modelos termina con una última transformación que genera el código fuente (o parte del mismo) de la aplicación final.

Existen distintos enfoques que utilizan los modelos como parte fundamental del diseño de software, por lo tanto en la bibliografía pueden encontrarse distintas siglas relacionadas. Basado en la categorización del libro Model-driven software engineering in practice (Brambilla, 2012) podemos decir que MDD es el nivel más bajo donde el desarrollo en sí, está dirigido por modelos en lugar de centrarse en la codificación. MDA (Model Driven Architecture) es una versión particular de MDD desarrollada por el Object Managent Group (OMG) que define una serie de modelos con distintos niveles de abstracción, partiendo de modelos independientes de la plataforma (PIM) hasta llegar mediante transformaciones a modelos específicos de cada plataforma (PSM) mediante los cuales es posible generar código fuente. MDE (Model Driven Engineering) es un superconjunto que engloba a MDD ya que incorpora otras características que van más allá del desarrollo incorporando todo el proceso de ingeniería de software. En este aspecto MDSE (Model Driven Software Engineering) como otras técnicas están englobadas dentro de MDE.

La utilización de MDD trae varias ventajas en el proceso de desarrollo de software entre las que es posible mencionar:

- Aprovechamiento de los modelos en etapas superiores mediante transformaciones
- Reducción del proceso de codificación
- Reducción del proceso de pruebas ya que el código generado automáticamente es menos probable que contenga errores una vez depurado el proceso de construcción automática.

- Al basarse en los modelos, el sistema queda documentado en todos los aspectos, facilitando el proceso de actualización o mejora.

La utilización de estas técnicas no elimina la necesidad de los programadores sino que el esfuerzo de programación es puesto en el desarrollo de transformaciones que puedan ser re-utilizadas en distintos sistemas. Mejorando los tiempos de desarrollo de nuevas aplicaciones y reduciendo notablemente los costos con una inversión inicial.

“MDA tiene el potencial para reducir ampliamente el tiempo de desarrollo e incrementar considerablemente la idoneidad de las aplicaciones; no lo hace mágicamente; sino proveyendo mecanismos por los cuales los desarrolladores pueden capturar su conocimiento del dominio y de la tecnología de implementación más directamente en una forma estandarizada y por la cual pueden utilizar este conocimiento para producir herramientas automatizadas que eliminan mucho del desarrollo de bajo nivel”. (Booch Grady, 2014)

Para especificar los modelos es necesario utilizar un lenguaje de modelado, y si bien se nombró a UML como el lenguaje estándar para la orientación a objetos, existe otra tendencia de especificar lenguajes propios del domino de la aplicación en cuestión. Este tipo de lenguajes son de propósito específico, al contrario de UML que es de propósito general y se denominan DSL (Lenguajes Específicos de Dominio) (Fowler Martin, 2010). Los DSL pueden especificarse de dos maneras:

- extendiendo UML utilizando sus mecanismos de extensión como estereotipos y valores etiquetados formando un perfil
- definiendo un lenguaje propio cuya especificación formal puede estar descripta en el lenguaje MOF (Meta-Object Facility). El MOF (OMG, OMG's MetaObject Facility, s.f.) es justamente un metalenguaje mediante el cual puede definirse la estructura de otros lenguajes de modelado. Por ejemplo UML tiene su especificación formal realizada en el lenguaje MOF. Esta especificación formal hace que se puedan definir relaciones entre los modelos para poder especificar transformaciones al utilizar la arquitectura MDA de la OMG.

Los DSLs permiten modelar un sistema utilizando una notación propia del dominio haciendo que los propios expertos sean capaces de especificar el sistema utilizando un lenguaje conocido por ellos, alejándose de los lenguajes generales para especificación de sistemas que requieren de conocimientos particulares. “Su uso puede ayudar a reducir la brecha entre el punto de vista de un experto del dominio del sistema software y su implementación” (France Robert, 2005)

El problema de los DSLs radica en que deben especificarse para el dominio en cuestión por lo que su aplicación estará limitada. Además desarrollar un DSL no es una tarea sencilla: “Desarrollar un DSL es complejo ya que requiere tanto conocimiento del dominio como experiencia en el desarrollo de lenguajes” (Mernik Marjan, 2005). En cambio la utilización de un lenguaje de propósito general permitirá especificar aplicaciones de cualquier dominio.

2.2 Metodologías de Modelado

Existen numerosas metodologías de modelado para aplicaciones web. Las mismas surgieron con el nacimiento del hipertexto (Landow, 1995) y fueron evolucionando a lo largo del tiempo incorporando distintas características y mejoras orientadas a la web. Los aspectos fundamentales que estas metodologías incorporan son la descripción de la información a visualizar, la navegación que se puede realizar para alcanzar dicha información y cómo será la visualización por parte del usuario final. Además muchas de las metodologías fueron diseñadas pensando en el enfoque MDD permitiendo derivar parte del código fuente partiendo de los modelos realizados.

La Figura 2 muestra las principales metodologías de modelado y sus años de publicación, así como también la relación existente entre ellas.

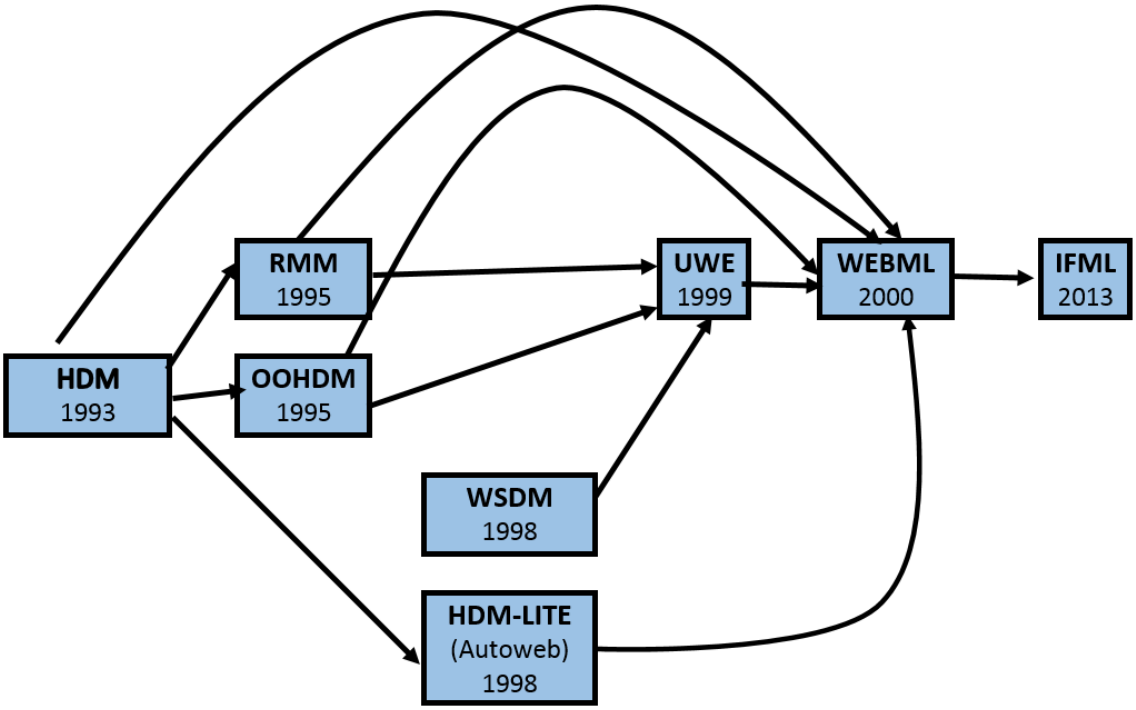

Figura 2. Principales metodologías de modelado para web

Cabe aclarar que la relación entre las distintas metodologías fue generada tomando la información provista por cada autor en la definición de su propia metodología. A continuación se listan brevemente cada una de las metodologías destacando sus principales características.

2.2.1 HDM

HDM (Hypertext Design Model) (Garzotto, Paolini, & Schwabe, 1993) surgió con el objetivo de crear un modelo que fuera de utilidad para realizar el diseño de una aplicación de hipertexto.

El proceso de diseño de una aplicación de hipertexto se divide en 2 partes: el authoring-in- the-large, que se refiere a la especificación y diseño de los aspectos globales y estructurales de la

aplicación (clases, interrelaciones), y el authoring-in-the-small, que se refiere al desarrollo del contenido de los nodos (obtención de la información desde una base de datos, y con que herramientas de desarrollo se programara, etc.). HDM se centra en el diseño a gran escala.

El proceso de desarrollo en HDM esta integrado por los siguientes pasos:

1. Identificar el grupo de entidades del mundo real y sus componentes. Las relaciones entre una entidad y sus componentes determinan la navegación estructural.
2. Identificar dentro de los objetos del mundo aquellos que tienen estructura y conexiones similares, es decir, los que son del mismo tipo.
3. Determinar el conjunto de colecciones de objetos y sus miembros. En este paso se diseñan los enlaces de aplicación derivados de las colecciones.
4. Se determina si el tipo de navegación de cada colección es mediante un índice o una visita guiada.

HDM es una herramienta de modelado conceptual, donde no hay detalle alguno de implementación, sino que permite construir una red de navegación de información. A partir de este modelo es posible derivar los link de hipertexto que deberá contener una aplicación para permitir navegar entre los distintos conceptos. Para ello se vale de distintos tipos de links:

- **De perspectiva**: no cambian el foco de atención del tema sino que muestra otro aspecto de la misma información.
- **Estructurales**: permiten interconectar distintas secciones de información correspondientes a la misma entidad, sirve por ejemplo para navegar información jerárquica o información dividida en varias páginas.
- **De aplicación**: son propios del dominio que se está modelando y permite especificar el tipo de la relación como una etiqueta dentro del mismo link. Por ejemplo el link “es Autor de” puede vincular un escritor con un libro.

2.2.2 RMM

RMM (Relationship Management Methodology) (Isakowitz, Stohr, & Balasubramanian, 1995) se define como un proceso de análisis, diseño y desarrollo de aplicaciones hipermedia. El modelo de datos está basado en HDM, pero a diferencia de este, RMM describe el diseño hipermedia y una metodología de desarrollo.

Los elementos principales de este método son el modelo Entidad-Relación usando para definir el modelo de datos y el modelo RMDM (Relationship Management Data Model) que permite crear distintas vistas sobre una entidad determinada agrupando las propiedades en lo que denomina slices (cortes) por lo tanto una entidad puede estar formada por uno o más cortes de información. RMDM también incluye seis primitivas para establecer la navegación como links unidireccionales, bidireccionales, condicionales, etc. Para visualizar la información maneja los conceptos de índices y visitas guiadas.

2.2.3 OOHDM

OOHDM (Object Oriented Hypermedia Design Methodology) (Schwabe & Rossi, Building hypermedia applications as navigational views of Information Models, 1995) es una metodología basada en HDM que se centra en el modelado con orientación a objetos.

Está dividida en 4 etapas (Schwabe & Rossi, An object oriented approach to Web-based applications design, 1998):

1. **Diseño Conceptual:** En esta etapa se modelan las clases de la aplicación con sus respectivas relaciones y jerarquías. Definiendo la semántica del domino de la aplicación.
2. **Diseño Navegacional:** Define la estructura de navegación de la aplicación, estableciendo uno o más contexto navegacionales. Sobre un mismo modelo conceptual pueden definirse varios modelos navegacionales para definir distintas vistas sobre el mismo dominio. Al igual que su predecesor (HDM) utiliza distintos tipos de conectores entre nodos: links, indices y tours guiados. El contenido de los nodos, es decir que información muestran, se especifican mediante un lenguaje de consulta sobre el modelo conceptual.
3. **Diseño de Interfaz Abstracta**: Define la interfaz de usuario utilizando objetos perceptibles (campos de texto, botones, etc). Los mismos son combinados definiendo un prototipo de la pantalla que incluye ademas como responder a eventos iniciado por el usuario y como se comunican los diferentes objetos navegacionales.
4. **Implementación**: En esta etapa se realiza un mapeo entre los objetos de la interfaz a objetos de implementación que ya dependen de la arquitectura elegida, como por ejemplo cliente-servidor.

2.2.4 WSDM

WSDM (Web Site Design Method) (De Troyer O.M.F., 1998) propone una metodología basada en el usuario, donde define un conjunto de usuarios potenciales del sitio web. Realiza una clasificación de dichos usuarios para adaptarse a distintos perfiles. Esta metodología está orientada a sitios web para consulta de información y no para aplicaciones web donde el foco está en administrar información.

El método está formado por cuatro fases:

- Modelado de Usuario, dividido en dos partes:
  - clasificación de usuario
  - descripción de clases de usuario
- Diseño Conceptual, que incluye el:
  - modelado de objetos
  - diseño navegación
- Diseño de la Implementación
- Implementación propiamente dicha.

2.2.5 HDM-Lite

HDM-Lite (Piero Fraternali, 1998) es una evolución de HDM diseñada específicamente para sitios web. Permite modelar un sitio web definiendo su modelo conceptual formado por diferentes esquemas que permiten:

- Describir la estructura de información que maneja la aplicación.
- Definir la forma en que se accede a la información descripta en el modelo anterior. Es decir define la navegación del sistema.
- Definir la apariencia de las páginas de la aplicación, pudiendo definir la distribución de la página y la ubicación de los controles.

Luego para lograr independencia de la interfaz, dinamismo y separación de conceptos, la metodología plantea una transformación del modelo conceptual a un modelo lógico donde cada uno de los esquemas se mapea a repositorios en una base de datos relacional.

El proceso de mapeo del modelo conceptual al modelo lógico, está soportado por la herramienta Autoweb que además basado en el modelo lógico permite generar páginas HTML y código Java. También esta herramienta cuenta con un editor gráfico para poder realizar el modelo conceptual inicial.

2.2.6 UWE

UWE (UML Web Engineering) (Baumeister Hubert, 1999) es una metodología de diseño web que busca utilizar a una notación estandarizada. Es por eso que se basa en UML creando una extensión al mismo mediante un perfil formado por distintos estereotipos tanto para modelar la navegación como la interfaz de usuario. También utiliza algunos valores etiquetados para modelar un tipo de navegación específico denominado navegación adaptativa, que se va cambiando la forma de navegar según el perfil de usuario.

UWE está basado en las metodologías OOHDM, RMM y WSDM (Koch, Knapp, Zhang, & Baumeister, 2008) y fue pensada dentro del marco de desarrollo dirigido por modelos, más específicamente siguiendo el estándar MDA de la OMG. Parte de un modelo independiente de la plataforma y mediante transformaciones semiautomáticas construye un modelo específico para la plataforma Java Server Pages (Oracle) del cual puede generarse código fuente.

Para realizar el modelo conceptual UWE utiliza el diagrama de clases UML sin extensiones de ningún tipo. En cambio, para el modelado de la navegación e Interfaz de usuario utiliza clases de UML estereotipadas

“El modelado de la navegación de una aplicación web comprende la construcción de dos modelos: el modelo de espacio de navegación y el modelo de estructura de navegación. El primero especifica que objetos pueden ser visitados mediante navegación en el sistema. Es un modelo a nivel de análisis. El segundo define como se accede a dichos objetos. Es un modelo a nivel diseño” (Koch Nora, 2002)

Para el modelo del espacio de navegación utiliza estereotipos para las clases de navegación y para los link de navegación directa. Para el modelo de estructura de navegación utiliza clases estereotipadas según la forma de mostrar la información teniendo índices, visitas guiadas, consultas y menús. También utiliza valores etiquetados para configurar algunas clases por ejemplo utiliza el tag sorted (ordenado) para indicar que la visualización de un clase se debe ordenar según las preferencias del usuario.

A nivel de presentación realiza el modelado de la interfaz de usuario utilizando clases estereotipadas que representan controles de la interfaz de usuario como por ejemplo links, textos, botones, formularios, etc.

Además la metodología requiere la construcción de otros modelos para especificar el comportamiento como por ejemplo diagramas de estados para definir el comportamiento de cada botón o diagramas de actividades para modelar tareas.

2.2.7 WebML

WebML (Web Modelling Language) (Ceri, Fraternali, & Bongio, 2000) es un lenguaje modelado de alto nivel para la especificación de aplicaciones Web. Está basado en HDM, HDM-Lite, RMM, OOHDM y Araneus (Atzeni Paolo, 1998) que es un repositorio de datos con administración web desarrollado en la Universidad de Roma.

WebML utiliza cuatro perspectivas:

1. Modelo Estructural: Expresa el contenido de los datos del sitio, en términos de las entidades y relaciones pertinentes. WebML no propone un nuevo lenguaje para el modelado de datos, sino que es compatible con notaciones clásicas como el modelo entidad relación y el diagrama de clases UML. Para hacer frente a la exigencia de expresar información redundante y calculada, el modelo estructural también ofrece una forma simplificada del lenguaje de consulta OQL (Object Query Language) por lo que es posible especificar la información derivada. OQL (UC San Diego Computer Science and Engineering, s.f.) es un lenguaje de consulta sobre objetos donde se puede acceder a las propiedades de un objeto mediante sintaxis de punto o realizar consultas complejas para obtener información relacionada.
2. Modelo de Hipertexto: Describe uno o más hipertextos que pueden ser publicados en el sitio. Cada hipertexto diferente define una vista de sitio llamada View Site, que a su vez consta de dos submodelos:
    - Modelo de Composición: Especifica qué páginas componen el hipertexto, y qué unidades de contenido constituyen una página. Existen seis tipos de unidades de contenido que se pueden utilizar para componer páginas: datos, multi-datos, índices, filtros, unidades scroller y directas. Las unidades de datos se utilizan para publicar la información de un solo objeto, mientras que los restantes tipos de unidades representan formas alternativas de navegar por un conjunto de objetos.

- Modelo de Navegación: Expresa las páginas y unidades de contenido que se unen para formar el hipertexto. Los enlaces son no contextuales, cuando conectan semánticamente páginas independientes, o contextuales, cuando el contenido de la unidad de destino del enlace depende del contenido de la unidad de origen. Los enlaces de contexto se basan en el esquema de estructura, ya que conectan las unidades de contenido cuyas entidades subyacentes están asociadas por las relaciones en el esquema de estructura.

3. Modelo de Presentación: Expresa el diseño y el aspecto gráfico de las páginas, independientemente del dispositivo de salida y de la interpretación del lenguaje, por medio de una sintaxis XML abstracta. Las especificaciones de presentación son, de una página específica o genérica. En el primer caso, establecen la presentación de una página específica e incluyen referencias explícitas al contenido de la página; en el segundo, se basan en modelos predefinidos independientemente del contenido específico de la página e incluyen referencias a elementos de contenido genérico.
4. Modelo de Personalización: Los usuarios y los grupos de usuarios se modelan explícitamente en el esquema de la estructura en forma de entidades predefinidas llamadas Usuarios y Grupos. Las características de estas entidades se pueden utilizar para almacenar contenido de un grupo específico o individual, como sugerencias de compras, listado de favoritos, y recursos para la personalización gráfica. Luego, se puede añadir expresiones declarativas de OQL al esquema de estructura, que definen el contenido derivado basado en los datos de perfil almacenados en las entidades de Usuario o de Grupo. Este contenido personalizado puede ser utilizado tanto en la composición de las unidades o en la definición de las especificaciones de la presentación. Además, reglas de negocios de alto nivel, escritas utilizando una sintaxis XML simple, se pueden definir para reaccionar a los eventos relacionados con el sitio, como clicks de usuario y actualizaciones de contenido. Las reglas de negocio se caracterizan por producir nueva información relacionada con el usuario (por ejemplo, el historial de compras) o actualizar el contenido del sitio (por ejemplo, la inserción de las nuevas ofertas que coincidan con las preferencias de los usuarios). Las consultas y reglas de negocio ofrecen dos paradigmas alternativos (uno declarativo y uno procedimental) para expresar y gestionar eficazmente los requisitos de personalización.

2.2.8 IFML

IFML (Interaction Flow Modeling Language) (OMG, Interaction Flow Modeling Language, 2013) nace en el 2013 como estándar surgido del OMG, está basado en WEBML.

“IFML soporta la descripción independiente de la plataforma de interfaces gráficas de usuario para aplicaciones desplegadas o accedidas en sistemas como computadoras de escritorio, laptops, PDAs (Personal Digital Assistant), teléfonos móviles y tablets. El foco principal está puesto en la estructura y comportamiento de la aplicación como la percibe el usuario final. El lenguaje de modelado también incorpora referencias a los datos y a la lógica de negocios que influencia la experiencia del usuario. Esto se logra respectivamente referenciando los objetos

del modelo de dominio que proveen el contenido que se muestra en la interfaz y las acciones que pueden ser desencadenadas al interactuar con la interfaz." (Brambilla Marco F. P., 2014)

"IFML no cubre el modelado de cuestiones de visualización (como layouts, estilo y look and feel), tampoco está diseñado para el modelado de juegos o aplicaciones altamente interactivas… está orientado a aplicaciones web de negocio que manejan datos." (Brambilla Marco B. S., 2014)

IFML permite especificar las vistas de la aplicación, su contenido, los eventos que se disponen en la interfaz y las acciones que dichos eventos producen en el sistema vinculándolos con la lógica de negocios. Al estar basado en MDD es posible generar en forma automática el código fuente para el front-end de las aplicaciones.

2.3 Trabajos Relacionados

Cabe destacar que las metodologías presentadas en la sección 2.2 tienen un importante punto de contacto con la presente tesis al estar todas orientadas al modelado de aplicaciones web, que incluyen en su mayoría capacidades de modelado de interfaz de usuario y generación automática de código fuente. Complementando lo presentado previamente, en esta sección se presenta una síntesis de trabajos académicos relacionados con la temática de la presente tesis, principalmente aquellos que hacen uso de componentes en MDD y generación de código mediante modelado de la interfaz de usuario.

Los trabajos relacionados serán catalogados según su enfoque principal, generando las siguientes categorías:

- Utilización de componentes en MDD
- Modelado de Interfaz de Usuario
- Modelado de Aplicaciones Web Móviles

2.3.1 Utilización de componentes en MDD

A continuación se presentan una serie de trabajos que incorporan el uso de componentes en MDD con distintos enfoques. En general se utilizan componentes tradiciones de software donde cada componente representa una funcionalidad con interfaces definidas para poder interactuar con otros módulos del sistema. Ninguno de los trabajos utiliza componentes de interfaz de usuario sino que el punto de coincidencia está dado en que cada componente en sí mismo tenga la capacidad de realizar transformaciones ya sea a un modelo de otro nivel como la capacidad de generar código fuente. Uno de los trabajos (Liu, Morisset, & Stolz, 2010) plantea la utilización de componentes en métodos formales mientras que el resto define DSLs para modelar dominios específicos. Otro autor (Johannes, 2011), se centra en la composición de modelos de funcionalidad reducida pero no permite configurar dichos modelos para adaptarse a distintas aplicaciones sino que son modelos auto contenidos con funcionalidades genéricas.

1. An Abstraction for Reusable MDD Components- Model-based Generation of Model-based Code Generators (Kulkarni Vinay, 2008)

Este trabajo utiliza técnicas de MDD para desarrollar generadores de código, esos generadores de código son organizados como bloques reutilizables con una distribución jerárquica.

Define componentes reutilizables que se pueden componer cada uno encapsulando una cuestión específica, definiendo en cada componente las transformaciones a realizar. A una misma clase es posible aplicarle distintos componentes por lo que la generación de código irá sumando los resultados de las transformaciones.

Define componentes para:

- generar bases de datos relacionales a partir de una clase
- agregar una tabla de auditoria a una clase particular con los métodos de grabación correspondiente
- generar código Java de las clases
- generar código JDBC (Java Database Connectivity), es decir el código para recuperar y almacenar datos en la base de datos relacional

La idea principal es separar las distintas partes de la generación de código, como componentes que pueden ser aplicados a una clase permitiendo parametrizar para cada clase como se quiere que sea la generación de código.

2. rCOS: Theory and Tool for Component-Based Model Driven Development (Liu, Morisset, & Stolz, 2010)

   Propone una arquitectura basada en componentes que permita:

   - describir las funcionalidades y comportamiento del sistema.
   - capturar la descomposición del sistema en componentes que cooperen e interactúen para cumplir con las funcionalidades generales del sistema.
   - soportar composición, coordinación y conexión de componentes describiendo la estructura jerárquica, dependencias y relaciones entre ellos.

   Este trabajo está orientado a especificación de interfaces por métodos formales por lo que se aleja de las notaciones gráficas centrándose en definir formalmente interfaces entre componentes para poder componer un sistema. Los componentes se utilizan en todas las etapas del proceso de desarrollo por lo que la conexión con MDD está dada en la transformación de componentes de análisis a componentes de diseño.

3. Reusable Specification Components for Model-Driven Development (Weiss K., 2003)

   Este trabajo propone la creación de componentes reutilizables para el dominio aeroespacial donde muchos de los sistemas de control tienen módulos comunes que están presente en la mayoría de los proyectos, más a allá de los detalles de implementación particulares. Hace hincapié especialmente en la reutilización indicando que la reutilización de código no es la más adecuada ya que necesita de muchas adaptaciones al problema particular, pero sí en cambio la reutilización se hace a un nivel mayor de abstracción utilizando componentes en

la etapa de modelado, luego se puede realizar una generación de código particular para el sistema en cuestión tomando una base pre-definida. Define los componentes desde un punto de vista de lo general a lo particular, pudiendo cada componente estar formado por componentes más pequeños. Hace hincapié en la construcción de componentes auto contenidos con interfaces bien definidas. Al estar enfocado a sistemas de control críticos también propone una definición formal de los componentes para que al combinarlos puedan realizarse validaciones automáticas y simulaciones.

4. Genie: Supporting the Model Driven Development of Reflective, Component-based Adaptive Systems (Bencomo Nelly, 2008)

   Propone la utilización de componentes configurables mediante instrucciones para reaccionar ante eventos para modelar sistemas adaptativos. Define dos lenguajes específicos de dominio uno para definir la estructura y otro para permitir la configuración de los componentes. Presenta además una herramienta que soporta la metodología que permite construir los modelos y generar código fuente y archivos de configuración XML. El problema principal que este enfoque presenta es la necesidad de la detección de conflictos entre las reglas de configuración.

5. Component Based Model-Driven Software Development (Johannes, 2011)

   Esta tesis busca incorporar las ventajas de desarrollo basado en componentes al desarrollo dirigido por modelos. Propone la utilización de modelos que tengan funcionalidades acotadas para que luego se puedan vincular con otros modelos realizando una composición para describir tareas más complejas. Desarrolla una técnica de composición que admite como entrada lenguajes de modelado tanto textuales como gráficos. La metodología está soportada por una herramienta denominada “Reuseware” que permite realizar la composición de modelos. Dicha composición puede realizarse siguiendo dos estilos arquitectónicos diferentes:

   1. Basada en separación de responsabilidades
   2. Jerárquica

   El foco principal de este trabajo está dado en la composición de modelos a partir de diversos lenguajes de modelado.

6. Generación Automática de Software para Sistemas de Tiempo Real: Un Enfoque basado en Componentes, Modelos y Frameworks (Alonso Diego, 2012)

   Este artículo presenta una metodología para el desarrollo de sistemas de tiempo real basada en componentes, donde el programador diseñe el sistema seleccionando y vinculando distintos componentes pre-definidos de un framework adecuado para el tipo de problema a solucionar. Los componentes seleccionados no son particulares de un lenguaje de programación específico sino que son modelos que mediante transformaciones automáticas pueden llevarse a distintas implementaciones.

2.3.2 Modelado de Interfaz de Usuario

La mayoría de las metodologías de modelado hipermedia presentadas en la sección 2.2 permiten realizar el modelado de la interfaz de usuario, basándose en el diseño de la pantalla mediante artefactos que representan controles y contenedores; haciendo que el diseño de la misma sea similar a desarrollar un prototipo de pantalla utilizando herramientas de maquetación. Además existen diversos trabajos académicos que se centran en el modelado de la interfaz de usuario. La principal relación con CBMDD de estos trabajos es la reutilización de componentes de interfaz de usuario. Los distintos autores concuerdan en que es posible modelar una aplicación tomando como base controles de interfaz de usuario pre-definidos, y que además esos componentes tengan la capacidad de permitir la generación de código fuente a partir de ellos. Entre los principales trabajos se encuentran:

7. PIM Tool: Support for Pattern-Driven and Model-Based UI Development (Radeke Frank, 2006)

   Este trabajo se basa en la utilización de modelos para representar la interfaz de usuario, pero además agrega el concepto de patrones para encapsular comportamientos comunes. De forma que al utilizar estos patrones el proceso de modelado sea más rápido, facilitando la reutilización, evitando el tener que combinar y configurar diversos modelos cuyo comportamiento ya está estandarizado. Para permitir la adaptación de esos patrones a distintas aplicaciones, los mismos pueden ser configurables mediantes distintas variables pre-definidas. También desarrollan una herramienta que permite realizar el modelado utilizando la metodología propuesta, identificando tareas y asignado patrones pre-definidos a cada tarea. El resultado final es un modelo de la interfaz de usuario pero no plantea la generación automática de código a partir de los modelos.

8. The UI Pilot: A Model-Based Tool to Guide Early Interface Design (Puerta Angel, 2005)

   Presenta una herramienta denominada UI Pilot que permite definir la interfaz de usuario mediante la realización de prototipos de pantalla del tipo wireframe. El prototipado está complementado con la vinculación a tareas y a un modelo de datos subyacente. Esta herramienta está pensada para realizar un diseño inicial de la interfaz que luego se irá modificando, siendo este el modelo inicial de un enfoque MDA. La herramienta permite exportar el trabajo a un formato de texto para que quede como documentación y un formato XML para que pueda ser tomado por otras herramientas para etapas posteriores del proceso de desarrollo.

9. The WUI-Toolkit: A Model-Driven UI Development Framework for Wearable User Interfaces (Witt Hendrik, 2007)

   Este trabajo define un framework basado en MDD para el desarrollo de interfaces de usuario específicas para productos que se pueden vestir (como por ejemplo relojes, pulseras, anteojos, etc.). Define un modelo de interfaz abstracto independiente de la plataforma de desarrollo que puede generar interfaces concretas para distintos dispositivos partiendo del

mismo modelo. Este framework se basa en la utilización de componentes reusables para reducir los esfuerzos de implementación. Permite diseñar interfaces que toman información del ambiente y del usuario. Dispone distintos componentes para poder definir interfaces independiente del dispositivo como por ejemplo: Información, Grupos, Listas de Selección, Disparadores de eventos e Ingreso de datos. Cada componente dispone de las funcionalidades necesarias para generar código fuente según el dispositivo destino. El trabajo presenta a modo de ejemplo una implementación concreta para dispositivos montados en la cabeza como por ejemplo anteojos.

2.3.3 Modelado de Aplicaciones Móviles

La mayoría de las metodologías de modelado presentadas en la sección 2.2 permiten también modelar aplicaciones web móviles además de aplicaciones de hipermedia tradicionales. Para aquellas que no lo soportan existen muchos trabajos académicos que extienden algunas de esas metodologías y las especializan para aplicarlas a dispositivos móviles. Pero hay otros trabajos que nacen pensando en los dispositivos móviles, enfocándose en algunas de sus características particulares y limitaciones. A continuación se muestran dos ejemplos de trabajos, uno se basa en el uso de MDD combinado con el diseño centrado en el usuario (Balagtas-Fernandez Florence, 2008) y el segundo (Thompson Chris, 2009) aplica MDD para generar prototipos tempranos y poder mejorar el consumo de energía en las aplicaciones móviles.

10. Model-Driven Development of Mobile Applications (Balagtas-Fernandez Florence, 2008)

    Este trabajo utiliza MDD complementado con técnicas de diseño centrado en el usuario para construir un modelo de alto nivel independiente de la plataforma, que genere de forma automática código fuente. La metodología está orientada a usuarios no expertos, sin conocimientos de programación ni de diseño de sistemas móviles. Los autores pretenden crear artefactos que permitan especificar para cualquier usuario sus propias aplicaciones en un alto nivel de abstracción. Este trabajo se encuentra en etapas tempranas de desarrollo con un prototipo de una herramienta que establece los módulos a desarrollar: herramientas para definir la interfaz de la aplicación, su navegación y definir los datos que la misma administra.

11. Optimizing Mobile Application Performance with Model-Driven Engineering (Thompson Chris, 2009)

    Este trabajo utiliza MDD no para modelar aplicaciones móviles completas y generar código funcional, sino para poder rápidamente establecer cuál será el consumo de energía de una aplicación móvil, cuestión importante cuando se utiliza por ejemplo un gran número de sensores del equipo móvil, como GPS, acelerómetro, etc. Plantea realizar un modelo conceptual donde los desarrolladores pueden especificar distintos tipos de módulos incluyendo:

    - Alto consumo de CPU (cálculos, notificaciones utilizando la ubicación actual, etc)

- Alto consumo de Memoria (visualización de imágenes, almacenamiento en disco, etc)
- Utilización de sensores (definiendo intervalos de chequeo de los mismos)
- Utilización de redes (envío y recepción de información por la red de datos)
- Agentes de Visualización (renderizado de imágenes en tres dimensiones)

El desarrollador define los módulos que su aplicación utilizará y mediante MDD se genera código fuente ejecutable en un dispositivo móvil, simulando las operaciones especificadas. Al ejecutar esta simulación se pueden correr distintas estadísticas con reportes de consumo y performance para que se pueda predecir y mejorar el consumo de la batería desde etapas tempranas del diseño.

Capítulo 3 – Propuesta

3.1 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: (a) estar bien diseñado desde el punto de vista de la usabilidad; (b) ser consensuado con el usuario final para mostrar la información que realmente le sea relevante, brindándole 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 tener interfaces necesariamente simples y directas. Diversas fuentes académicas proveen principios importantes (W3C, Mobile Web Application Best Practices, 2010) (W3C, Mobile Web Best Practices 1.0, 2009) (Krug, 2006) (Nielsen & Budio, 2012) 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.

Al centrarse en aplicaciones que administran datos, se puede observar que tienen un diseño similar donde la mayoría incorpora formas de visualizar, buscar y editar la información. Esto permite definir una serie de pantallas base 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 para poder realmente especificar el comportamiento de la interfaz es necesario establecer una forma de definir qué datos se desean visualizar en cada una de estas. Por ejemplo, en:

- un listado, se debe definir los datos a mostrar, como están ordenados, que sucede al pulsar sobre un dato.
- un menú, se deben definir las distintas opciones y a que pantalla conducirán cada una de ellas.
- una búsqueda, se deben definir cuáles son los filtros que el usuario podrá completar.
- 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 utilizado en muchas metodologías de modelado como por ejemplo: OOHDM (Object Oriented Hipermedia Design Method) (Schwabe & Rossi, 1998) y UWE (UML Based Web Engineering) (Koch, Knapp, Zhang, & Baumeister, 2008). Otras metodologías utilizan diagramas de entidad relación como WebML (Web Modeling Language) (Ceri, Fraternali, & Bongio, 2000) y RMM (Relationship Management Methodology) (Isakowitz, Stohr, & Balasubramanian, 1995). Para la construcción de la metodología planteada en la presente tesis se elige el diagrama de clases ya que forma parte de UML. 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 deseada.

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 pantalla sea representada por un componente y cada componente pueda configurarse mediante un modelo de datos previamente definido.

3.2 Utilización de Componentes en el Desarrollo Dirigido por Modelos

Tomando como premisa la generación automática del código fuente completo de una aplicación y además tomando en cuenta la necesidad de simplificar el proceso de modelado, se crea una nueva metodología. Esta metodología utiliza componentes configurables pre-establecidos de interfaz de usuario para definir el comportamiento del sistema. La metodología está enfocada al desarrollo puntual de aplicaciones por lo que se enmarca dentro de MDD según la clasificación establecida en el la sección 2.1.

La utilización de componentes en el proceso de desarrollo de software es una técnica muy bien establecida en la industria de software (Heineman & Councill, 2001). Los componentes son piezas de software pre-definidas con un propósito definido. “El desarrollo de software basado en componentes, permite que la construcción del software se realice mediante el ensamblado de bloques pre-fabricados configurables y de evolución independiente” (Keller & Reinhard, 1998).

Los componentes son excelentes para la re-utilización y son confiables ya que una vez que son testeados y su comportamiento fue verificado, se puede utilizar en un gran número de aplicaciones sin modificación. “Los beneficios de los componentes de software incluyen la reutilización y la interoperabilidad, entre otros” (Adler, 1995).

La mayoría de los frameworks de programación modernos incluyen componentes pre-definidos 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 lo que se denomina en esta tesis como CBMDD (Component Based Model Driven Development).

La OMG en la especifición de MDA, nombra al desarrollo basado en componentes como una posibilidad para reducir el trabajo de modelado tomando desiciones comunes en componentes pre-definidos: “La madurez del desarrollo basado en componentes, donde un middleware provee un conjunto de servicios y en la cual las decisiones arquitectónicas se toman una sola vez para un gran número de proyectos, creando sistemas similares. Estas decisiones se implementan en las herramientas, procesos de desarrollo, templates, bibliotescas de programación y generadores de código. En ese contexto es posible que un desarrollador cree un PIM que sea completo. El desarrollador puede especificar el comportamiento directamente en el modelo utilizando un lenguaje de acción. Esto hace que el PIM sea computacionalmente completo, es decir que contenga toda la información necesaria para producir el código fuente de la aplicación.” (OMG, MDA Guide Version 1.0.1, 2003)

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:

1. Facilita el desarrollo, prueba e implementación de las transformaciones ya que una vez funcional, la transformación se reutiliza en todas las aplicaciones posteriores.
2. Facilita el modelado al tener bloques pre-fabricados que pueden configurarse para adaptarse a distintas aplicaciones.
3. Reduce los tiempos de desarrollo, reutilizando componentes y transformaciones previamente desarrollados y probados.
4. Aumenta la calidad del producto final por tratarse de bloques ya testeados.

Los pasos a seguir para generar componentes configurables en CBMDD son los siguientes:

1. Definir bloques comunes que las aplicaciones puedan utilizar y empaquetarlos en un componente.
2. Agregar configuración a cada componente teniendo en cuenta el brindar toda la información necesaria para que se pueda generar código fuente partiendo de dicha configuración.
3. Realizar las transformaciones desde y hacia los componentes. Es decir, si los componentes basan su configuración en un modelo anterior, entonces partiendo de dichos modelos se deben generar automáticamente componentes pre-configurados que puedan ser útiles en la mayoría de las aplicaciones y luego adaptarlos según las necesidades del usuario. Por otro lado una vez configurados los componentes se debe generar el código fuente de la aplicación partiendo de ellos.

3.3 Metodología para el Modelado de Aplicaciones Web Móviles

En esta sección se define una metodología que emplea componentes configurables en MDD para modelar aplicaciones web móviles. Los componentes definen distintas partes de la interfaz de usuario del sistema y su configuración se basa en un primer modelo de datos de la aplicación.

Esta metodología se plantea inicialmente 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 se basa en componentes, cada uno de ellos va a representar una pantalla que se mostrará al usuario. Por otra parte 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 (W3C, Mobile Web Best Practices 1.0, 2009), 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.

A pesar de estar enfocada principalmente a aplicaciones web móviles, esta metodología puede aplicarse también a aplicaciones web tradicionales donde no se requieran distribuciones de pantallas demasiado complejas.

3.3.1 Pasos de la Metodología

El esquema general de pasos se encuentra representado en la Figura 3.

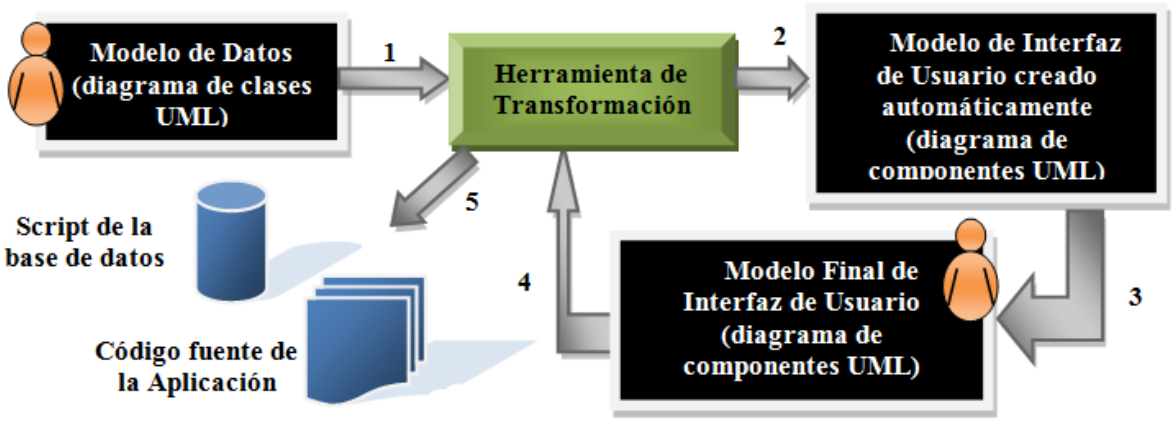

Figura 3. Pasos de la Metodología

Los pasos representados en la figura 3 son descriptos brevemente a continuación:

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 versión inicial del modelo de interfaz de usuario, incluyendo la navegación básica entre los distintos componentes.

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.

5 Como resultado de la transformación anterior se genera el script de la base de datos y el código fuente de la aplicación.

Tal como puede observarse en la figura 3 la intervención del usuario se produce al inicio y en el paso 3, el resto se realiza en forma automática.

3.3.2 Construcción de un Perfil de UML

Para definir un nuevo lenguaje de modelado, tal como se mensionó anteriormente en el capítulo 2 existen dos opciones, crear un perfil de UML ó definir un nuevo lenguaje basado en MOF. Se opta por la opción de crear un perfil de UML para basarse en la utilización de estándares y además permitir realizar los modelos con las herramientas de modelado existentes en el mercado.

“Es precisamente a la hora de describir el modelo de la plataforma y las reglas de transformación entre modelos cuando los Perfiles UML pueden desempeñar un papel muy importante. Si usamos Perfiles UML para especificar el modelo de una plataforma, eso nos garantiza que los modelos derivados serán modelos consistentes con UML. De hecho, la principal regla para aplicar MDA con éxito es usar en lo posible estándares (modelos estándar y Perfiles UML de lenguajes o plataformas estándar).” (Fuentes & Vallecillo, 2004)

Los dos diagramas utilizados en la metodología están basados en diagramas tradicionales de UML extendidos mediante estereotipos y valores etiquetados. Las extensiones realizadas se definen mediante el perfil de UML de la Figura 4. Todos los nuevos estereotipos definidos heredan de una de las metaclases 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” (Rossi, Schwabe, Olsina, & Pastor, 2008).

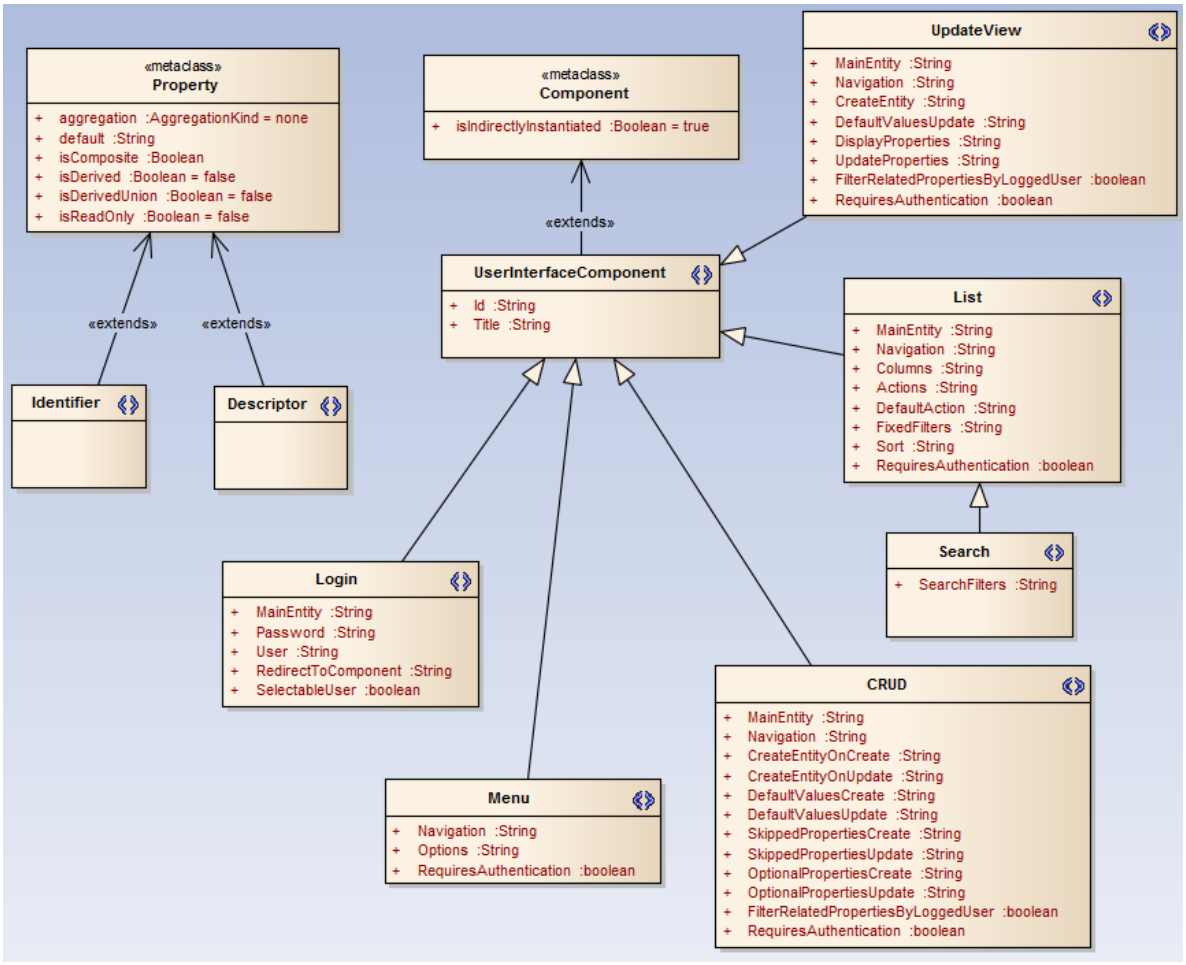

Figura 4. Perfil UML para la metodología CBMDD

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 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 debe tener una única propiedad con el estereotipo `Identifier` (la misma es obligatoria)

4. Una clase debe tener una única propiedad con el estereotipo `Descriptor` (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`

3.3.3 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 extensiones, las cuales se describen a continuación.

3.3.3.1 Estereotipos para propiedades especiales

Por cada clase se deben identificar dos propiedades especiales el identificador y el descriptor.

- El identificador es una propiedad numérica que va a identificar unívocamente un objeto de esa clase. Esta propiedad se transformará luego en la clave primaria de la tabla que almacenará los objetos de la clase y es señalada mediante el estereotipo `<identifier>`
- El descriptor es una propiedad de tipo texto que también identifica unívocamente un objeto de esa clase pero de forma entendible por el usuario. Por ejemplo el nombre de una persona, el título de un libro. Tener identificada esta propiedad servirá para referirse al objeto cuando deba utilizarse en un control de la interfaz de usuario. Por ejemplo si se dispone de una clase de países y en la interfaz se muestra un control para la selección de un país, el usuario verá en pantalla los nombres de los países si la propiedad nombre fue marcada como descriptor de dicha clase. Esta propiedad se señala mediante el estereotipo `<descriptor>`

3.3.3.2 Clases que representan enumeraciones

En algunas aplicaciones el modelo de datos puede contener clases que representan valores limitados de determinado objeto, por ejemplo, los distintos estados por los que atraviesa una factura, un listado de acciones posibles, etc. A estas clases es necesario distinguirlas del resto ya que al momento de generar el código fuente de la aplicación deben tener un tratamiento especial, debido a que son clases de sólo lectura donde los objetos ya son definidos dentro del propio diagrama. Por lo tanto al definir una clase de este tipo, deberán también definirse todos los valores de dicha enumeración. UML ya cuenta con un estereotipo denominado `<enumeration>` que permite identificar estas clases particulares, por lo cual el mismo se utiliza sin modificaciones.

3.3.3.3 Tipos de Dato Especiales

Con la finalidad de mejorar la experiencia del usuario al utilizar la aplicación y teniendo en cuenta que la metodología está enfocada principalmente a dispositivos móviles, se definen dos nuevos tipo de datos:

- `phoneNumber`: es un texto que representa un número telefónico. Esto permite que al momento de generar la interfaz de usuario para visualizar dicho campo se pueda crear un link para llamar directamente a dicho número o enviar un mensaje de texto. Esta es una funcionalidad recomendada por el W3C en su guía de buenas prácticas para aplicaciones web móviles en la sección 3.5.6 de dicho documento: "Permitir el click para llamar en números telefónicos" (W3C, Mobile Web Application Best Practices, 2010).
- `address`: este tipo de dato se corresponde con una ubicación geográfica lo que brinda la posibilidad de aprovechar las características de geo-posicionamiento presente en la mayoría de los dispositivos móviles, ya sea mediante GPS o triangulación de redes WIFI. Tener esta propiedad identificada permite crear interfaces de usuarios más ricas, por ejemplo para tomar automáticamente la ubicación del dispositivo o para mostrar un mapa de ubicación de la dirección almacenada. Pero al agregar esta funcionalidad debe tenerse en cuenta que no siempre puede estar disponible o que simplemente el usuario no quiera utilizarla.

3.3.4 Modelo de Interfaz de Usuario

En este modelo se diseñaran las distintas pantallas que conformaran el sistema, determinando además, cuál es el flujo de navegación entre las mismas. 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 quiere modelar se crearon estereotipos que se aplican a los componentes de UML para tipificarlos. Los estereotipos disponibles son: `Login`, `List`, `Search`, `Menu`, `CRUD` (create, read, Update, Delete) y `UpdateView`.

Cada tipo de componente tiene definido una serie de valores etiquetados que permite configurar su comportamiento para adaptarlo a la aplicación que se está modelando. Algunos de estos valores etiquetados son comunes para la mayoría de los componentes y otros son particulares para cada uno de ellos.

Los valores etiquetados comunes presentes en la mayoría de los componentes son:

- `Id`: es un valor que identifica unívocamente un componente. Es de utilidad cuando un componente es referenciado por otro, por ejemplo al diseñar la navegación.
- `Title`: es la descripción textual del componente. La misma será mostrada al usuario al momento de generar la interfaz de usuario.
- `RequiresAuthentication`: es un valor booleano que indica si para poder acceder al componente el usuario debe primero estar logueado en el sistema o no. Cuando está

seteado en verdadero y un usuario no logueado quiere acceder al componente, será redireccionado a la pantalla de login de forma automática.

- `Navigation`: este tag define una serie de links que forman parte de la barra de navegación que estará presente en todos los componentes.
- `MainEntity`: se utiliza para establecer la clase del modelo de datos en la cual está basado principalmente el componente. Por ejemplo, en una pantalla de consulta de información (si bien puede consultarse información proveniente de distintas clases) siempre existirá una clase principal que será la base de la consulta. El definir esta clase principal también facilita la configuración, ya que al referirse a una propiedad de la clase por defecto, se podrá escribir directamente su nombre sin necesidad de especificar primero a clase pertenece.

A continuación se explicarán cada uno de los componentes con sus correspondientes valores etiquetados:

- `Login`: este componente sirve para autenticar un usuario para poder utilizar el sistema. Sus valores etiquetados son:
  - `Id`
  - `Title`
  - `MainEntity`
  - `RedirectToComponent`
  - `User`
  - `Password`
  - `SelectableUser`
  - `Navigation`

  `User` y `Password` define las propiedades de la clase (configurada en `MainEntity`) donde serán chequeadas las credenciales del usuario.
  `RedirectToComponent` tiene el `Id` del componente donde el usuario será redireccionado después de un logueo exitoso. El usuario autenticado será accesible desde el resto de los componentes mediante una variable especial denominada LOGGEDUSER; esta variable puede ser utilizada para configurar los distintos componentes. El componente puede configurarse para permitir seleccionar el usuario en lugar de tener que escribirlo utilizando el tag `SelectableUser`. Esta característica es útil cuando se tienen pocos usuarios en el sistema y especialmente adecuada para dispositivos móviles ya que minimiza el ingreso de texto, tarea compleja en pequeños dispositivos en especial en aquellos con teclado reducido. El tag `Navigation` es opcional y puede utilizarse para especificar links para llevar a cabo acciones antes de loguearse, como por ejemplo un link para poder crear un nuevo usuario sino se dispone de uno.

- `List`: Este componente representa un listado de información que se visualiza como una grilla mostrando información del modelo de datos. Sus valores etiquetados son: `Id, Title, MainEntity, Navigation, RequiresAuthentication,`

`FixedFilters`, `Columns`, `AditionalInformationLine`, `Sort`, `Actions` y `DefaultAction`.

- `FixedFilters`: define los filtros que se aplicarán a la consulta para recuperar la información que se visualiza en el listado. Estos filtros se aplican sobre las propiedades de la clase definida en el tag `MainEntity` pudiendo además acceder a sus clases relacionadas mediante notación de objetos.
- `Columns`: Define cada una de las propiedades que se visualizarán en el listado, según la interfaz generada se mostrarán como diferentes columnas o no. En el caso de los dispositivos móviles como el uso de tablas no es recomendado, si se definen varias columnas, las mismas serán concatenadas en único texto a visualizar. Además de poder indicar propiedades a visualizar, puede también aplicarse diversas funciones para mostrar datos calculados, las mismas se encuentran detalladas en la sección 3.3.4.1.
- `AditionalInformationLine`: Al presentar un listado en un dispositivo móvil debido al tamaño de pantalla reducido es habitual mostrar la información en filas con dos renglones, es decir un primer renglón con la información más importante y un segundo renglón con información adicional. Este enfoque es muy utilizado para todo tipo de listado incluso en los sistemas operativos móviles, en los menús internos, visualización de mensajes, etc. Mediante este tag es posible configurar las propiedades a visualizar en dicha línea adicional de información.
- `Sort`: define las propiedades por las cuales el listado va a ordenarse. Puede indicarse tipo de orden ascendente o descendente mediante las palabras reservadas `ASC` y `DESC`.
- `Actions`: representa las acciones posibles relacionadas con cada fila del listado. Está formada por un conjunto de links que pueden llevar a distintos componentes con funcionalidades diferentes. Por ejemplo se puede configurar: un link para visualizar información, otro para borrar y uno para modificar. De forma automática el componente destino recibirá el `id` del objeto sobre el cual se quiere realizar la acción, este dato está relacionado con el identificador de la clase configurada como entidad principal (`MainEntity`) del listado. Al tener varias opciones luego de pulsar sobre la fila se muestra al usuario un submenú con las posibles acciones a realizar.
- `DefaultActions`: cuando al pulsar sobre una fila del listado sólo es posible realizar una acción, se utiliza este valor etiquetado. De esta forma se redireccionará al usuario al componente deseado sin necesidad de mostrar un menú de acciones. Este es el enfoque recomendado para aplicaciones móviles para incrementar la usabilidad.

- `Search`: Este componente es similar al componente `List` agregando la capacidad de incorporar filtros que, en lugar de estar predefinidos, son ingresados por el usuario al momento de utilizar la aplicación, permitiendo realizar consultas dinámicas sobre los

datos. El resultado de esa consulta se visualiza en forma de listado. Sus valores etiquetados son: `Id`, `Title`, `MainEntity`, `Navigation`, `RequiresAuthentication`, `FixedFilters`, `Columns`, `AditionalInformationLine`, `Sort`, `Actions`, `DefaultAction`, `FilterRelatedPropertiesByLoggedUser` y `SearchFilters`. Como puede apreciarse respecto al componente `List` agrega solo dos etiquetas:

- `SearchFilters`: que representa los filtros que serán completados por el usuario en tiempo de ejecución. Los filtros de búsqueda se configuran mediante la propiedad sobre la cual se realizará la búsqueda y el tipo de filtro a aplicar. Los tipos de filtro disponibles son:
  - `SingleSelection`: se utiliza para clases relacionadas y permite seleccionar solo un elemento para el filtro.
  - `MultipleSelection`: se utiliza también para clases relacionadas pero permitirá seleccionar una o más opciones.
  - `FreeText`: es una búsqueda por texto libre, puede aplicarse tanto a campos del tipo texto como a clases relacionadas donde se busca en su descriptor.
  - `BooleanType`: solo aplicable a tipos de dato booleanos. Permite filtrar por verdadero o falso.
  - `DateRange`: para búsquedas sobre campos del tipo fecha, muestra una fecha “desde” y otra “hasta” para permitir el filtro por rangos.
- `FilterRelatedPropertiesByLoggedUser`: permite acotar las opciones de búsqueda a aquellas relacionadas con el usuario logueado para aquellas clases relacionadas con este.

- `Menu`: permite configurar un menú de opciones que puede usarse como punto de partida de la aplicación. Sus valores etiquetados son: `Id`, `Title`, `Navigation`, `RequiresAuthentication y Options`. Para configurar las opciones del menú se definen una serie de links que apuntan a otros componentes mediante el tag `Options`.

- `CRUD`: este componente es el responsable del manejo de datos, permitiendo la creación (**C**reate), visualización (**R**ead), actualización (**U**pdate) y eliminación (**D**elete) de un registro de una tabla. Sus valores etiquetados son:
  - `Id`
  - `Title`
  - `Navigation`
  - `MainEntity`
  - `DefaultValuesCreate`
  - `DefaultValuesUpdate`

- `SkippedPropertiesCreate`
- `SkippedPropertiesUpdate`
- `CreateEntityOnCreate`
- `CreateEntityOnUpdate`
- `OptionalPropertiesCreate`
- `OptionalPropertiesUpdate`
- `RequiresAuthentication`
- `FilterRelatedPropertiesByLoggedUser`

Como puede verse este es el componente con más cantidad de opciones de configuración debido a que permite configurar el comportamiento por separado para la creación y para la actualización de datos. Sin embargo, la mayoría de sus valores etiquetados son opcionales lo que hace rápida su configuración si el sistema no requiere restricciones sobre el tratamiento de los datos en el momento de alta o modificación. Del listado de valores etiquetados los obligatorios son los 4 primeros y en los dos últimos se debe asignar un valor booleano, con lo cual pude notarse que es rápido el seteo en este caso.

Todos estos valores etiquetados hacen que combinados puedan adaptarse a distintas situaciones según el problema que se desee modelar.

Si hay información que debe ser completada en el objeto pero no solicitada al usuario ya que se completa de forma automática se utilizan los valores etiquetados `DefaultValuesCreate` o `DefaultValuesUpdate` para un proceso de alta o modificación respectivamente. Esto permite por ejemplo dejar asentado el usuario que crea o modifica el registro y la fecha de actualización del mismo.

Si hay propiedades de una clase que no deben ser completadas en el proceso de creación del registro se utiliza el valor etiquetado `SkippedPropertiesCreate`. Análogamente, si algunas propiedades deben permanecer inalteradas en el proceso de modificación de datos se utiliza el valor etiquetado `SkippedPropertiesUpdate`.

También en los procesos de alta y modificación pueden existir propiedades que son obligatorias y otras opcionales. Por defecto todas las propiedades son requeridas salvo que se indique en forma explícita que las mismas serán opcionales utilizando los valores etiquetados: `OptionalPropertiesCreate`, `OptionalPropertiesUpdate`.

El valor etiquetado `FilterRelatedPropertiesByLoggedUser` se utiliza cuando el objeto que estamos actualizando tiene relación con otras clases que dependen del usuario logueado, permitiendo que al momento de la edición sólo puedan seleccionarse aquellos valores relacionados con el usuario logueado actualmente en el sistema.

El componente está preparado también para crear un registro adicional en otra tabla cuando se crea o modifica un registro de la tabla principal, lo que permite por ejemplo

crear un esquema de datos con una tabla de log separada. Para configurar la creación de registros adicionales se utilizan los valores etiquetados `CreateEntityOnCreate` y `CreateEntityOnUpdate`.

En tiempo de ejecución el comportamiento del componente está regido por un parámetro recibido en el link que causa la carga de dicha página. El link contiene un parámetro denominado `Action` que puede recibir los siguientes valores: `C`,`R`,`U`,`D`, `UD`, donde cada una de las letras indica una acción posible:

- `C`: Creación, alta de nuevos registros
- `R`: Lectura, visualización de la información
- `U`: Actualización, modificación de la información
- `D`: Eliminación, borrado del registro
- `UD`: configura el componente tanto para Actualización como para Eliminación permitiendo las dos acciones indistintamente. En todas las operaciones salvo en el alta de nuevos registros también se recibe un parámetro llamado `ObjectId` que contiene el `id` del registro sobre el cual se realizará la operación

- `UpdateView`: Este componente está diseñado para realizar actualización parcial de la información, permitiendo configurar aquellas propiedades que podrán ser modificadas por el usuario y aquellas que se mostrarán como de sólo lectura. De esta forma un registro puede irse completando parcialmente a medida que va cambiando de estado dentro del sistema, permitiendo visualizar información cargada previamente que ya no es editable. Sus valores etiquetados son:
  - `Id`
  - `Title`
  - `Navigation`
  - `MainEntity`
  - `DisplayProperties`
  - `UpdateProperties`
  - `CreateEntity`
  - `DefaultValuesUpdate`
  - `RequiresAuthentication`
  - `FilterRelatedPropertiesByLoggedUser`

  Como puede apreciarse la mayoría de los valores etiquetados son idénticos a los del componente CRUD. El valor etiquetado `CreateEntity` cumple la misma funcionalidad que `CreateEntityOnUpdate` del componente CRUD, simplemente cambia el nombre ya que este componente siempre realiza operaciones de actualización. Se agregan dos valores etiquetados nuevos: `DisplayProperties` para especificar aquellas propiedades que serán visualizadas como solo lectura y

`UpdateProperties` para definir aquellas propiedades que podrán ser modificadas por el usuario.

3.3.4.1 Funciones de Consulta de Datos

Al configurar los componentes se hace referencia al modelo de datos utilizando la notación de objetos. Es decir que usando la sintaxis de puntos es posible acceder a clases relacionadas y a sus propiedades. Pero este enfoque muchas veces no es suficiente. Por tal motivo y con el fin de brindar mayores posibilidades durante la configuración de componentes, se incorporan diversas funciones para la manipulación de datos. Esto permite realizar un diseño de base de datos complejo, ya que luego la información puede ser recuperada mediante consultas avanzadas.

A continuación se muestra el detalle de configuración de componentes con funciones que permiten realizar una configuración avanzada de los mismos. Durante la especificación de la sintaxis de configuración se utilizan las llaves para datos requeridos y corchetes para parámetros opcionales.

Las funciones para consulta de datos son:

- `Sum`: esta función permite sumar una propiedad de una clase en un subconjunto de registros que cumplan determinada condición. La sintaxis de esta función es la siguiente:

```
Sum ({clase}.{propiedad}, [condición/es]);
```

- `Count`: cuenta la cantidad de registros que cumplen con la condición especificada dentro de una determinada clase. Su sintaxis es:

```
Count ({clase}, [condición/es]);
```

- `Exist, Not Exist`: estas dos funciones retornan un valor booleano para chequear si existe o no algún registro con la condición especificada. Su sintaxis es:

```
Exist ({clase}, {condición/es});
Not Exist ({clase}, {condición/es});
```

- `Min / Max / Avg`: permite recuperar valores máximos, mínimos o promedio de una propiedad de una clase. Pueden aplicarse opcionalmente condiciones para reducir el conjunto de datos sobre los cuales realizar la operación.

```
Min ({clase}.{propiedad}, [condición/es]);
Max ({clase}.{propiedad}, [condición/es]);
Avg ({clase}.{propiedad}, [condición/es]);
```

3.3.4.2 Variables Especiales de sólo Lectura

La metodología contiene algunas variables predefinidas con un significado especial que sirven para configurar los componentes de mejor forma. Son variables de sólo lectura ya que recuperan su valor del contexto pero no permiten su modificación explícitamente. Esas variables son:

- `LOGGEDUSER:` Contiene el id del usuario logueado en el sistema. Esta variable es accesible en todos los componentes y puede utilizase si el sistema modelado contiene un componente del tipo `Login` y el componente en el cual se utiliza está configurado como que requiere autenticación. El id de usuario será guardado luego de una autenticación exitosa en el sistema mediante el componente `Login`. Puede utilizarse por ejemplo para mostrar información filtrada para el usuario logueado en el sistema.
- `NOW`: Permite obtener la fecha y hora actual. Se puede utilizar para guardar información automática en una propiedad por ejemplo para dejar registrada la fecha y hora de actualización de un registro.
- `TODAY`: Permite acceder al día en curso (sin la hora). Puede ser útil por ejemplo para filtrar registros del día.
- `BACK`: Es una variable que se utiliza en la navegación para indicar que se quiere regresar al componente que invocó la llamada actual. Es decir volver al componente anterior. Es de utilidad esta palabra reservada ya que muchas veces un componente es reutilizado e invocado desde distintas partes del sistema y esta variable permite almacenar el componente que lo invocó para poder regresar a él mediante un link en la navegación.
- INVOKEID: esta variable hace referencia al ID del ítem seleccionado en una grilla de una pantalla previa que invocó al componente actual. Se utiliza en los componentes `List` o `Search` para poder realizar filtros por defecto con valores seleccionados en una pantalla anterior. Esto permite realizar un esquema de maestro detalle donde en un primer listado se selecciona un ítem, y en se va a otro listado filtrando otro objeto por el ítem seleccionado en la primera grilla.

3.3.4.3 Links

Para configurar la navegación y las acciones en los distintos componentes se utilizan links. Los links se representan en forma de función con distintos parámetros:

```
Link( {texto}, {destino}, [parametroCRUD], [TeclaAcceso]);
```

Donde:

- `texto:` es un literal que se pone entre comillas que indica el texto que se mostrará al usuario como enlace.
- `destino`: indica el destino del link que puede ser: un componente, una URL externa o la variable especial `BACK`. Cada tipo de destino tiene una configuración particular:
  - Componente: en el caso de que el link lleve a otro componente directamente se coloca el identificador de dicho componente, es decir el texto configurado en el campo `Id` del componente destino.
  - ULR externa: para indicar que el link lleva a un sitio externo se utiliza la función:

    `URL({dirección web})`

    Donde `{dirección web}` es la URL destino comenzando con http://

  - `BACK`: Si se utiliza esta variable especial, el link automáticamente permitirá regresar al componente que invocó la pantalla que actualmente se está visualizando.

- `ParametroCRUD`: este es un parámetro opcional para la función Link pero es requerido si el destino es un componente del tipo CRUD ya que obligatoriamente debe especificarse el parámetro que configura su comportamiento.

- `TeclaAcceso`: opcionalmente se puede asignar una tecla de acceso rápido para que en el caso de estar desarrollando un sitio web para dispositivos básicos, pueda agregarse la opción de navegar directamente a dicho link pulsando una tecla numérica. Los valores que admite este parámetro son numéricos del 0 al 9. El uso de teclas de acceso es recomendado por el W3C en su guía de mejores prácticas para la web móvil (W3C, Mobile Web Best Practices 1.0, 2009).

Para dar mayor flexibilidad al sistema es posible también configurar links opcionales, es decir links que se muestren o no al usuario, dependiendo si se cumple determinada condición sobre los datos. El formato de la función de links opcionales agrega un parámetro adicional obligatorio para indicar el o los filtros a aplicar.

```
OptionalLink( {texto}, {destino},[parametroCRUD], [TeclaAcceso],
{Condicion/es});
```

Las condiciones se aplican sobre el modelo de datos permitiendo el uso de funciones, variables especiales, agrupamiento mediante paréntesis y los operadores de comparación de la Tabla 1.

Tabla 1. Operadores de comparación

Símbolo	Tipo de comparación
==	Igualdad
<	Menor
>	Mayor
<>	Desigualdad
>=	Mayor o igual
<=	Menor o igual

Para concatenar condiciones se utilizan los operadores lógicos AND y OR.

Mediante el uso de links opcionales se pueden configurar link para distintos roles de usuario, haciendo que cada usuario vea un menú personalizado según su nivel de acceso. Esto facilita la configuración del sistema sin necesidad de duplicar componentes e incluso permite adaptarse a distintas formas de configurar la seguridad según el modelo de datos que se quiera utilizar. A continuación se muestran tres ejemplos diferentes para configurar roles en el sistema y definir links opcionales según el rol del usuario logueado:

1. Uno más roles, directamente asignados en la clase del usuario Este esquema requiere que la clase del usuario tenga propiedades del tipo booleano para indicar la pertenencia o no del usuario a cada grupo. El ejemplo de la Figura 5 muestra una clase Usuario con tres propiedades booleanas que indican roles diferentes.

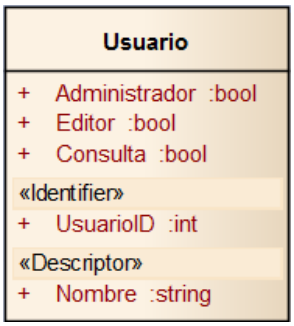

Figura 5. Clase con asignación de roles mediante propiedades booleanas

Tomando el ejemplo de la Figura 5 si se desea configurar un link para que sea sólo visible por un usuario con el rol administrador la condición que se debe configurar es la siguiente:

```
LOGGEDUSER.Adminitrator == true
```

Puede notarse que nos referimos directamente a la variable especial LOGGEDUSER que identifica el usuario logueado y luego utilizando sintaxis de punto es posible acceder a las propiedades de la clase que corresponde al objeto del usuario logueado.

Si quiere mostrarse el link tanto para administradores como para usuarios de consulta la condición es:

```
LOGGEDUSER.Adminitrator == true OR LOGGEDUSER.Consulta == true
```

2. Rol único, con clave foránea de clase relacionada: Si un usuario puede tener un único rol asignado, el esquema más habitual será el de tener una tabla con los roles predefinidos que se representa mediante una clase de tipo enumeración. Luego en la clase de usuario se tendrá una propiedad del tipo de la clase enumeración, tal como ejemplo de la Figura 6.

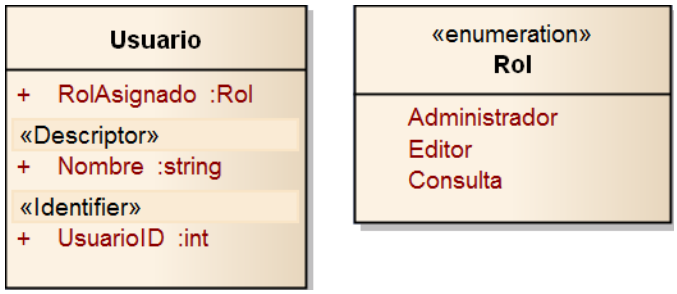

Figura 6. Esquema de asignación de seguridad con un único rol por usuario

Con este esquema para restringir la visibilidad de un link sólo para usuarios administradores la condición es:

```
LOGGEDUSER.RolAsignado == Administrador
```

3. Uno o más roles, en clase relacionada: Si a cada usuario puede asignársele más de un rol y se dispone de una tabla de roles entonces la relación se realiza en una clase separada como en el ejemplo de la Figura 7.

Figura 7. Esquema de seguridad para múltiples roles en clase relacionada

En este caso la condición para que el link solo sea visible para el rol administrador es la siguiente:

```
RolesUsuario.Usuario == LOGGEDUSER AND RolesUsuario.Rol == Administrador
```

3.3.4.4 Configuración de los valores etiquetados

Cada valor etiquetado tiene una configuración particular con una sintaxis específica que debe ser respetada. A continuación se detalla la forma de configuración de cada uno de los valores etiquetados, agrupando aquellas cuya configuración es equivalente.

- `Actions, Options, Navigation`
  Puede contener uno o más links separados por coma. La sintaxis para representar los links fue desarrollada en la sección 3.3.4.3.

- `DefaultAction`
  Se configura mediante un único link. El texto asignado al mismo no es relevante ya que no se mostrará en la interfaz.

- `Columns, AditionalInformationLine`
  Se configura mediante los nombres de una o más propiedades de la clase setead como clase principal. En caso de incluir más de una propiedad, las mismas se separan con coma. También se admite la sintaxis de punto para acceder a clases relacionadas.

- `CreateEntity, CreateEntityOnUpdate, CreatEntityOnCreate`
  Lista de nombres de clases separadas por coma.

- `DefaultValuesCreate, DefaultValuesUpdate`
  Se configura mediante la propiedad a la cual se desea asignar un valor y se explicita dicho valor que se tomará en forma automática. La propiedad se especifica de forma completa de la siguiente forma:

  ```
  Clase.Propiedad = valorAsignado
  ```

  Donde `valorAsignado` puede tomar información de dos fuentes:

- De un valor de una enumeración, en cuyo caso se especifica con sintaxis de punto:

  `Clase.ValorEnumeracion`.

- De una variable del entorno, pudiendo ser una de tres opciones:
  - `LOGGEDUSER`
  - `NOW`
  - `TODAY`

- `DisplayProperties, UpdateProperties, OptionalPropertiesCreate, OptionalPropertiesUpdate, SkippedPropertiesCreate, SkippedPropertiesUpdate`
  Es un listado de propiedades separado por comas, de la entidad principal del componente al cual pertenece. En este caso como todas las propiedades son de la entidad principal no se utiliza sintaxis de puntos sino que directamente se configura mediante el nombre de la propiedad.

- `FilterRelatedPropertiesByLoggedUser, RequiresAuthentication, SelectableUser`
  Son propiedades booleanas por lo tanto se configuran directamente como `true` para valores verdaderos o `false` para valores falsos.

- `FixedFilters`
  Es una expresión que representa uno o más filtros. Los filtros se expresan mediante notación de punto con:

  `Clase.Propiedad OperadorComparacion valorFiltro`

  Donde:
  - `OperadorComparacion` se refiere al tipo de comparación para el filtro que puede ser cualquiera de los operadores especificados en la Tabla 1.
  - `valorFiltro` puede ser una variable de entorno o un valor de una enumeración.

  Además las condiciones pueden concatenarse con los operadores lógicos `AND` y `OR`. También pueden agruparse con paréntesis para dar prioridad.

- `Password, User`
  Su configuración se realiza mediante el nombre de una única propiedad relacionada con la entidad principal del componente.

- `RedirectToComponent`
  La configuración se realiza mediante el `Id` del componente al cual será redireccionado el usuario.

- `SearchFilters`
  Los filtros de búsqueda se configuran mediante el nombre de la propiedad a la cual aplicar el filtro y seguido de un espacio el tipo de filtro a aplicar. Para configurar más de un filtro de búsqueda los mismos se separan por comas. Los tipos de filtro de búsquedas fueron detallados en la explicación del componente `Search` en la sección 3.3.4.

- `Sort`
  La especificación de las propiedades por las cuales se ordenan los listados se realiza mediante el nombre de la propiedad seguido de un espacio y la dirección del ordenamiento que puede ser ASC para orden ascendente o DESC para orden descendente.
- `Title`
  Es un texto libre que indica el título del componente.

3.3.5 Transformaciones

Tal como se mencionó anteriormente esta metodología prevé dos transformaciones, ambas automáticas ya que los modelos en los cuales se basan tienen toda la información necesaria para generar el modelo destino.

La metodología solo utiliza modelos independientes de la plataforma donde se describen componentes en forma genérica que mediante transformaciones pueden generar código fuente en distinas plataformas. La OMG en su documento de especificación de MDA preevee este caso: "Hay contextos en los cuales el PIM provee toda la información necesaria para la implementación y no hay necesidad de agregar marcas o utilizar perfiles adicionales para generar código…. La herramienta interpreta el modelo directamente o transforma el modelo directamente en el código del programa " (OMG, MDA Guide Version 1.0.1, 2003)

3.3.5.1 Transformación de modelo a modelo

La primera transformación toma como entrada el modelo de datos y genera de forma automática una primera versión del modelo de interfaz de usuario. De esta forma se reduce considerablemente el tiempo de modelo ya que el único modelo completo que realiza el usuario es el modelo de datos. El modelo de interfaz solo necesitará ser configurado según las necesidades de la aplicación a modelar, lo que puede incluir la creación de algún componente adicional.

La transformación desde el modelo de datos al modelo de interfaz de usuario se basa en las clases definidas en el primer modelo exceptuando aquellas clases del tipo enumeración. Por cada clase se realizan automáticamente las siguientes acciones:

1. Se genera un componente del tipo `CRUD` que va a permitir administrar los objetos de dicha clase.
2. Se genera un componente del tipo `List` que va a permitir visualizar el listado de objetos de dicha clase. En cada ítem de la lista se genera una acción con un link hacia el componente `CRUD` creado en el punto 1 para editar o eliminar el ítem seleccionado. También en la barra de navegación del componente `List` se agrega un link para acceder al componente `CRUD` del punto 1 pero en este caso configurado para crear un nuevo registro.

3. Se genera un componente del tipo `Menu` con links a los distintos componentes del tipo `List` creado en el punto 2.
4. Se configura la navegación de todos los componentes creados:
    - desde los componentes CRUD para poder regresar al listado desde el cual se accedió al mismo.
    - desde los componentes `List` para poder regresar al `Menu` generado en el punto 3.

De esta manera 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.

3.3.5.2 Transformación de modelo a código

Para obtener una aplicación funcional, la transformación final consta de dos partes, la creación del script de base de datos y la creación del código fuente.

1. Basado en las clases del modelo de datos se genera un script en ANSI SQL. Este script incluye:
    - Creación de la base de datos con el nombre del proyecto.
    - Una tabla con cada clase y enumeración incluyendo todas las propiedades configuradas. La propiedad marcada como `Identifier` se configura como clave primaria de dicha tabla y para la propiedad marcada como `descriptor` se crea un índice del tipo `unique` para evitar valores duplicados.
    - En el caso en que las propiedades de la clase sean del tipo de una clase relacionada, se crean las claves foráneas correspondientes.
    - En las tablas correspondientes a clases del tipo enumeración se insertan las filas con los valores definidos en dicha enumeración.
2. La segunda parte de la transformación toma ambos modelos, el modelo de datos y el de interfaz de usuario y genera el código fuente de la aplicación. Este paso dependerá de la plataforma destino para la cual se quiera generar el código fuente. El modelo dispone de toda la información necesaria para realizar un código 100% funcional, dicha información se encuentra en cada uno de los valores etiquetados según el tipo de componente. La información disponible para la creación automática del código fuente es la siguiente:
    - Con el modelo de datos se pueden generar para cada clase, una clase equivalente el en lenguaje de programación elegido así como también los métodos para acceso a la base de datos para recuperar, modificar y eliminar registros de cada tabla.

- Según el tipo de componente es la pantalla que se debe generar en el sistema y cada componente tiene en sus valores etiquetados la información necesaria para la creación del código. En la sección 3.3.4 se detallaron cada uno de los valores etiquetados para los componentes y su utilización.

Capítulo 4 – Implementación

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. Por otro lado fue necesario desarrollar una herramienta que pueda llevar a cabo las transformaciones propuestas en la metodología. Es por eso que se decidió crear una herramienta que soporte tanto el modelado como las transformaciones.

La herramienta es una aplicación Web desarrollada en C# bajo el framework Microsoft ASP.NET v4 utilizando Web Forms (Microsoft, ASP.NET Web Forms, s.f.). 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 (XML Metadata Interchange).
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.
- Disponibilizar clases para generar fácilmente código fuente en cualquier lenguaje.

El desarrollo fue realizado en múltiples capas. Las mismas pueden verse en la Figura 8 donde puede apreciarse que existe una capa para la interfaz de usuario llamada UI que corresponde al sitio web que será el que el diseñador utilice para realizar el modelado. Luego cuenta con una capa Data que se encarga de todo el acceso a la base de datos, una capa Process que encapsula todo el manejo del modelo, incluyendo las transformaciones e interoperablidad. Por último una capa Common donde se definen clases compartidas entre las distintas capas, como las vistas y las clases correspondientes a las tablas del modelo de datos.

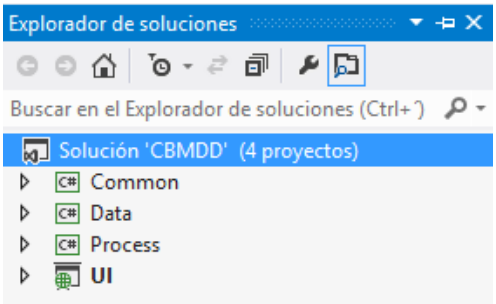

Figura 8. Capas de la aplicación

La aplicación cuenta con una base de datos donde almacena la información de los proyectos y de los distintos modelos. Está pensada para que varias personas puedan colaborar en un mismo proyecto, por lo tanto un proyecto puede vincularse con uno o más usuarios. Un proyecto debe tener un único modelo de datos pero puede tener más de un modelo de interfaz de usuario para el caso de que se deseen generar interfaces para distintos dispositivos (por ejemplo se podría generar un modelo para la interfaz móvil y un modelo para una interfaz de configuración web). A continuación se explica brevemente el propósito de cada una de las tablas:

- Users: Contiene la información de los usuarios que utilizarán el sistema incluyendo su clave de acceso.
- ProjectUsers: Vincula a los usuarios con los proyectos. Los usuarios podrán visualizar y modificar solo aquellos proyectos a los cuales estén vinculados.
- Projects: Contiene cada uno de los proyectos. Cada proyecto implica el modelado de un sistema particular. También incorpora información de fechas de actualización de cada uno de los diagramas y de las transformaciones para poder brindar al usuario información de trazabilidad y dar aviso de cuando un diagrama queda desactualizado por un cambio en una etapa anterior.
- Entities: Contiene cada una de las clases y enumeraciones del modelo de datos, adicionalmente hay un campo que guarda información del script SQL generado para dicha entidad para poder generar luego un script de actualización en el caso de que se modifique un campo.
- EnumerationValues: En el caso de que la entidad se defina como enumeración entonces esta tabla contendrá los distintos valores asignados a dicha enumeración. También incorpora información para generar scripts de SQL de actualización.
- EntityProperties: En el caso de que la entidad se defina como una clase tradicional entonces esta tabla contendrá cada una de las propiedades de dicha clase incorporando marcas para identificar aquellas propiedades del tipo `Descriptor` y `Identifier` que son los dos estereotipos para las propiedades definidos en el modelo. También incorpora información para generar scripts de SQL de actualización.
- DataTypes: Esta tabla contiene los distintos tipos de datos que pueden asignarse a las propiedades de las clases incorporando además un campo para indicar como nombrar dicho tipo de datos al momento de generar el script de la base de datos en SQL y otro campo para poder identificarlo al momento de importar o exportar el modelo hacia otras herramientas.
- ComponentDiagrams: Contiene los diagramas de interfaz de usuario de la aplicación que se está modelando. Generalmente sólo se contará con un único diagrama que a su vez es generado en forma automática por la primera transformación a partir del modelo de datos.
- Components: Contiene los distintos componentes de cada diagrama, indicando para cada uno si fue generado en forma automática y además con que clase del diagrama de datos se relaciona. Es decir para aquellos componentes que tienen un tag `MainEntity` la relación con dicha entidad se guarda en esta tabla en el campo `RelatedEntityId`.

- ComponentDetails: Almacena la configuración de cada componente, guardando el valor asignado a cada uno de los valores etiquetados.
- TaggedValueTypes: Esta tabla almacena los distintos tipos de valores etiquetados.
- ComponentTypes: Esta tabla almacena los distintos tipos de componentes.
- TaggedValueByComponent: Esta tabla asigna los valores etiquetados que deben configurarse en cada tipo de componente, indicando además un orden y si son requeridos o no, para completar el modelo.
- TaggedValuesControlTypes: Para facilitar el desarrollo se han realizado controles de usuario reutilizables para agrupar aquellos valores etiquetados que se configuran de la misma forma permitiendo que se utilicen los mismos controles. Por ejemplo, hay valores etiquetados del tipo texto libre, otros del tipo booleano o controles más complejos como selección de propiedades, asignación de filtros, creación de columnas, etc. Se agruparon los valores etiquetados según la Tabla 2.

Tabla 2. Tipos de controles para configurar los valores etiquetados

Tipo de Control	Valor Etiquetado
Boolean	SelectableUser
	RequiresAuthentication
	FilterRelatedPropertiesByLoggedUser
Columns	Columns
Default Property Value	DefaultValuesCreate
	DefaultValuesUpdate
Filters	FixedFilters
Multiple Entity Property Selection	AditionalInformationLine
	SkippedPropertiesCreate
	SkippedPropertiesUpdate
	DisplayProperties
	UpdateProperties
	OptionalPropertiesCreate
	OptionalPropertiesUpdate
Multiple Entity Selection	CreateEntity
	CreateEntityOnUpdate
	CreatEntityOnCreate
Multiple Links	Navigation
	Options
	Actions
Search Filters	SearchFilters
Single Component Selection	RedirectToComponent
Single Entity Property Selection	User
	Password
Single Link	DefaultAction
Sort Properties	Sort
Text	Title

La Figura 9 muestra el esquema completo de la base de datos con las relaciones entre las distintas tablas.

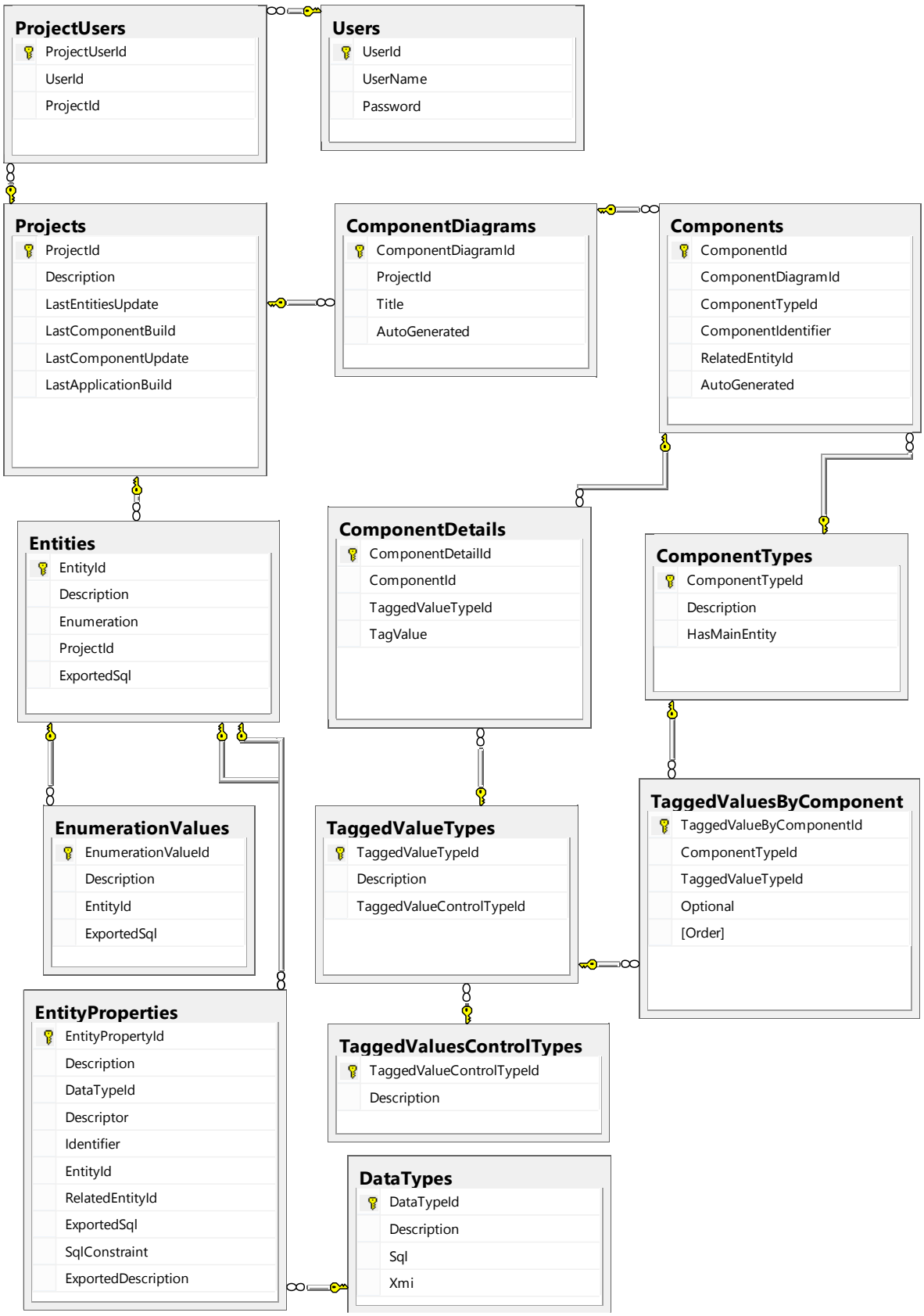

Figura 9. Esquema de base de datos de la aplicación

4.1 Soporte para el modelado

La herramienta permite modelar la aplicación de forma rápida, siguiendo los pasos y requerimientos de la metodología. El modelado no se realiza en forma gráfica sino que se buscó la mejor forma para facilitar la configuración de los modelos que es lo importante en la metodología. Se permite modelar múltiples proyectos por cada usuario, por lo tanto luego del login, el usuario podrá acceder a sus proyectos y seleccionar en cual desea trabajar, tal como muestra la Figura 10.

Figura 10. Selección del proyecto de trabajo

Luego de seleccionado el proyecto se accede a un menú con las distintas acciones que se pueden realizar sobre el proyecto. La Figura 11 muestra el menú de un proyecto con las distintas acciones posibles:

- Configurar el modelo de datos definiendo la clases
- Generar los componentes partiendo del modelo de datos
- Modificar y agregar nuevos componentes
- Generar el código fuente y script SQL
- Configurar los usuarios con acceso al proyecto

Figura 11. Menú de acciones del proyecto

4.1.1 Modelo de Datos

Para cada proyecto se debe configurar el modelo de datos, para ello el usuario cuenta con la pantalla de la Figura 12 donde puede crear clases y enumeraciones o modificar las ya creadas.

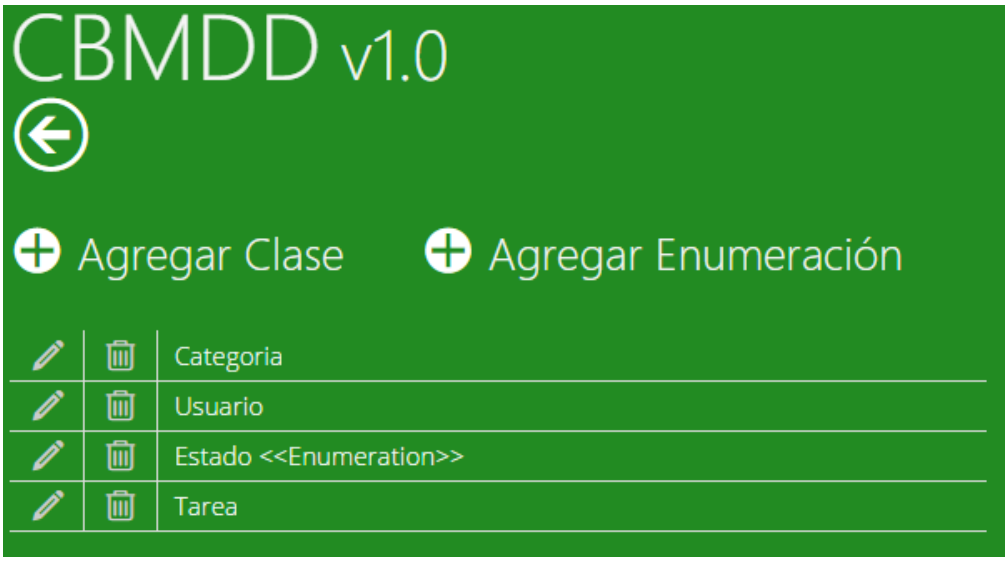

Figura 12. Modelo de datos de un proyecto

Al agregar tanto una clase como una enumeración se exigen que se ingresen los nombres de los campos `Identifier` y `Descriptor` requeridos por la metodología. De forma automática se establece el campo `Identifier` como una propiedad numérica y el campo `Descriptor` como una propiedad de tipo texto.

En el caso de:

- una clase, pueden agregarse las propiedades adicionales deseadas seleccionando el tipo de datos.
- una enumeración, se exige ingresar al menos un valor para la misma.

La Figura 13 muestra del lado izquierdo la pantalla de creación de una clase y del derecho la pantalla de creación de una enumeración.

Figura 13. Izquierda: creación de una clase – Derecha: creación de una enumeración

4.1.2 Modelo de Interfaz de Usuario

El modelo de interfaz de usuario se realiza mediante un diagrama de componentes. El sistema está preparado para que por cada proyecto pueda crearse más de un diagrama en el caso de que se quieran generar interfaces diferentes para distintos entornos. Por ejemplo, se podría diseñar una interfaz de configuración para la web de escritorio y otra para la web móvil. Cada una de estas interfaces se configura en un diagrama de componentes diferentes y luego al momento de generar el código fuente se selecciona la plataforma destino para cada uno de ellos.

Un modelo de interfaz de usuario está formado por varios componentes, cada uno de ellos configurado con los valores etiquetados correspondientes. La Figura 14 muestra la pantalla de visualización del diagrama de componentes donde un listado detalla cada uno de los componentes que incluye, permitiendo editar sus valores etiquetados o la información básica del componente. Al dar de alta un componente, primero se configura su información básica (ver Figura 15), es decir su identificador, tipo de componente y entidad relacionada (aquella que se indica en el tag `MainEntity` del modelo). Esta configuración inicial es necesaria ya que determina que configuración se debe realizar y que valores puede tomar esa configuración.

Figura 14. Componentes de un Diagrama de Interfaz de Usuario

Figura 15. Creación de un componente

Para realizar la configuración de los componentes se han creado controles de ayuda según el tipo de información que se necesita ingresar. Existen 13 tipos de controles diferentes que son suficientes para poder configurar los 27 valores etiquetados que tiene el modelo. La relación entre los valores etiquetados y el tipo de control para poder configurarlos fue detallada en la Tabla 2. Al ingresar a configurar un componente se mostrarán los valores etiquetados con los controles adecuados para configurarlos. A modo de ejemplo, se muestran en la Figura 16 y la Figura 17 los controles de ayuda para poder configurar el orden de una lista y la barra de navegación respectivamente.

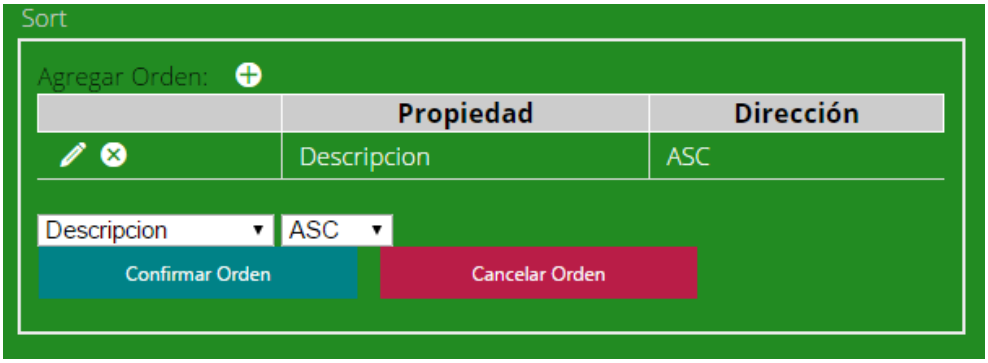

Figura 16. Ayuda de configuración para un campo de ordenamiento

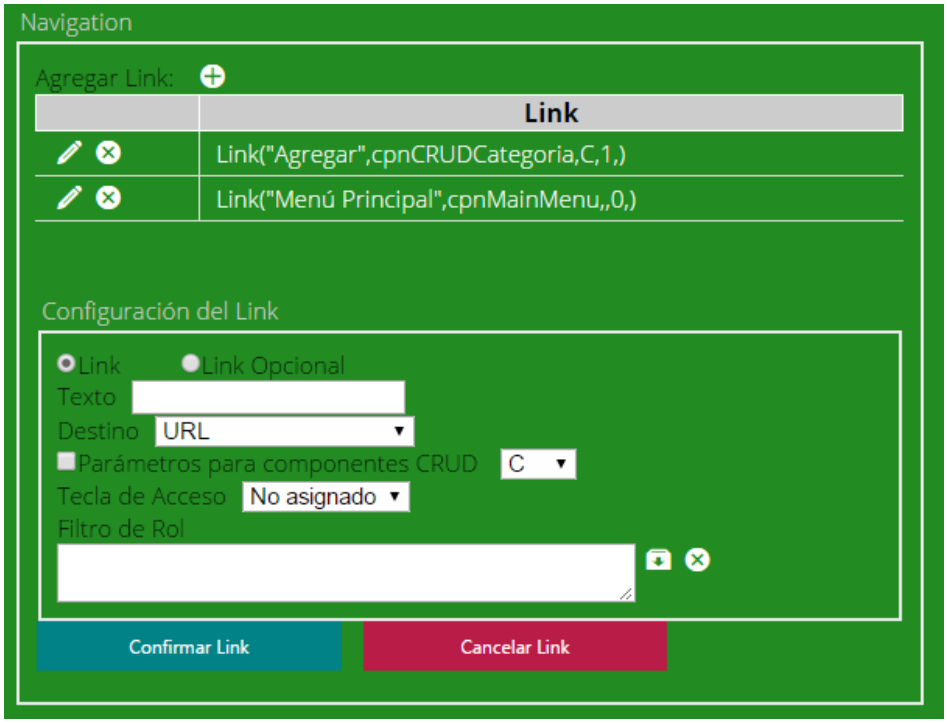

Figura 17. Ayuda para la configuración de links para el valor etiquetado `Navigation`

En el ejemplo de la Figura 18 se muestra la configuración de un componente del tipo login. Donde para cada valor etiquetado se muestra un control diferente:

- `Title`: Campo de texto libre.
- `RedirectToComponente`: Listado de componentes disponibles.

- `User` y `Password`: Listado de propiedades de la clase configurada como relacionada del componente (MainEntity).
- `SelectableUser`: Al ser un campo booleano se muestra un control para poder poner el verdadero o falso.
- Navigation: control para poder agregar uno o más links. Similar al de la Figura 17.

Figura 18. Valores etiquetados de un componente del tipo Login

4.1.3 Trazabilidad entre modelos

La herramienta almacena la fecha y hora de última modificación de cada modelo para poder notificar al usuario si hay cambios pendientes que no se impactaron en los modelos siguientes. Como se mencionó anteriormente la herramienta cuenta con transformaciones automáticas y por lo tanto detecta que las transformaciones deben volver a realizarse para impactar las modificaciones realizadas.

De esta forma se puede notificar al usuario cuando:

- **Los componentes están desactualizados**: Si se realiza algún cambio en el modelo de datos los componentes quedarán desactualizados en los siguientes casos:
    1. Al agregar o eliminar alguna clase.
        - Si se agrega una clase se deben generar los componentes `CRUD` y `List` correspondientes y actualizar el menú para apuntar al listado.
        - Si se elimina una clase, los componentes `CRUD` y `List` se deben eliminar, así como también el link correspondiente desde el menú.
    2. Al modificar el `Descriptor` de alguna clase, cambia la información de los listados ya que por defecto se muestra dicha propiedad en el listado.

- **El código está desactualizado**: Cualquier cambio tanto en el modelo de datos como en el modelo de componentes hacen que el código quede desactualizado. El script SQL queda desactualizado ante cualquier cambio del modelo de datos por lo tanto al volver a genera el código el mismo se volverá a crear si se detectan dichos cambios.

La Figura 19 muestra cómo se notifica al usuario cuando se detecta que se deben volver a generar los componentes y/o el código fuente.

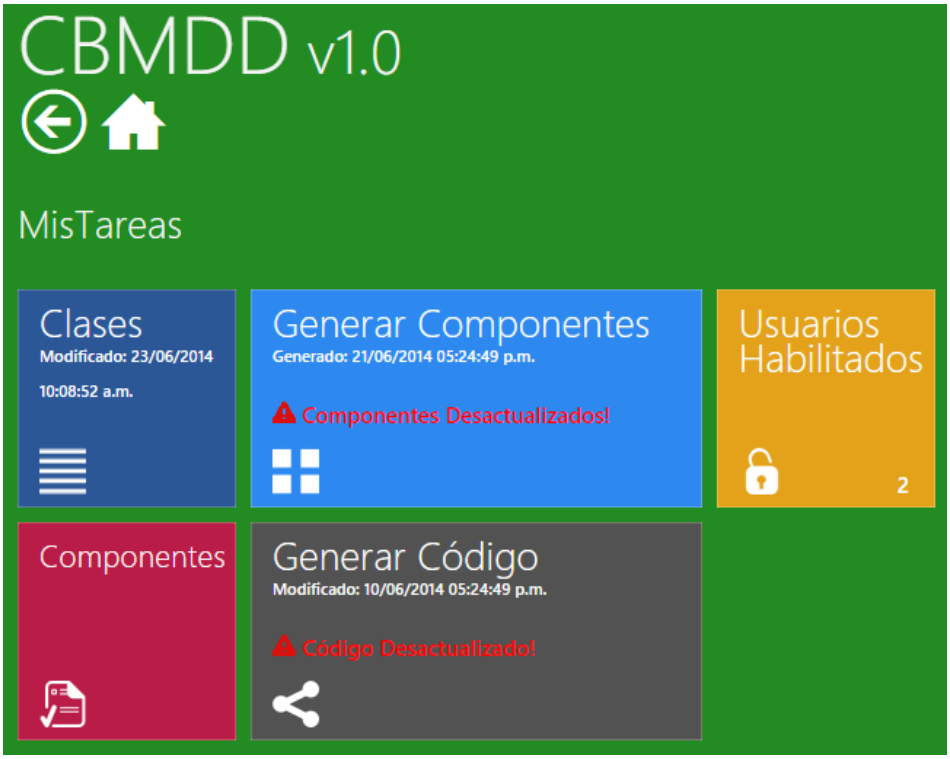

Figura 19. Notificaciones para trazabilidad de los modelos

Debido a que esta metodología genera el código fuente completo, los cambios se hacen directamente sobre los modelos volviendo a generar el código, de forma que la trazabilidad se hace en un único sentido.

Para el caso del script de la base de datos es posible que se agreguen campos a una aplicación que ya está en uso y por lo tanto el modelo está preparado para generar un script con las modificaciones sobre la base en lugar de realizar el script completo de creación.

4.1.4 Interoperabilidad

Si bien la herramienta permite realizar tanto el modelado como las transformaciones, el diseñador puede querer utilizar su herramienta de modelado UML habitual para realizar los modelos o simplemente para visualizarlos en forma gráfica. Por eso se agregó al sistema la posibilidad de importar y exportar los modelos.

La IEEE define a la interoperabilidad como la "Habilidad de un Sistema o producto de trabajar con otros sistemas o productos sin esfuerzo especial por parte del cliente. La interoperabilidad es posible mediante la implementación de estándares" (IEEE, 2010).

El estándar definido para el intercambio de modelos UML es el XMI (OMG, XML Metadata Interchange (XMI), Version 2.4.2, 2014). XMI es un formato de intercambio de metadatos mediante XML, en el mismo se representan modelos UML, tanto su definición como las modificaciones sobre los mismos.

El estándar definido por la OMG se desarrolló para permitir la exportación de información y meta información (donde se incluyen los diagramas UML) creado por una herramienta Case y permitir abrirlo con otra herramienta. Sin embargo este estándar no cuenta con una definición detallada en algunos aspectos del armado del XMI, como ser los parámetros obligatorios de la cabecera o la utilización de namespaces estándares. Cada herramienta Case genera su propio documento y frecuentemente no es posible el intercambio de modelos entre ellas, ya que, la que lo esté importando traduce parcialmente el diagrama exportado y en ocasiones no llega a iniciar la importación por necesitar algún detalle en la cabecera o similar, esto es un requisito que adiciona la herramienta en sí ya que el estándar no lo solicita obligatoriamente. A continuación se detallan algunas inconsistencias en la exportación con distintas herramientas:

- Cabecera: El XMI al ser un archivo XML define su estructura mediante el namespace que tiene asignado en la cabecera del documento. Hay herramientas que asignan el namespace del estándar de UML, por ejemplo en el caso de Enterprise Architect (Sparx Systems, 2015) (xmlns:uml=http://schema.omg.org/spec/UML/2.1). Sin embargo otras herramientas como lo es ArgoUML (Odutola & van der Wulp, 2010) tienen sus propios namespaces.
- Cuerpo del XMI: En el desarrollo del XMI se generan los elementos del diagrama con un nombre propio como `packagedElement` pero indicándole en la declaración de dicho elemento el tipo de dato que corresponde, seteándole la propiedad `xmi:type`. Por ejemplo en el caso de las clases, lo que hace Enterprise Architect es crear un nuevo elemento `packagedElement` con la propiedad `xmi:type="uml:Class"` además le agrega como parámetro el `xmi:id`, asignándole un id único, y las propiedades `name` y `visibility`. Por ejemplo:

```
<packagedElement xmi:type="uml:Class"
xmi:id="EAID_ED159BEF_E9FF_476a_99CE_4C837A8165D2"
name="Company" visibility="public"/>
```

Sin embargo otras herramientas arman el XMI de otra forma, por ejemplo en el caso de ArgoUML en vez de crear elementos nuevos genera directamente elementos del tipo `UML:Class`, donde no asigna la propiedad `xmi:type` por ejemplo:

```
<UML:Class xmi.id - '-64--88-56-1-4b0595f1:13eed195ce6:-
8000:0000000000000A5E' name - 'Clase1' visibility -
'public' isSpecification - 'false' isRoot - 'false' isLeaf
-'false' isAbstract - 'false' isActive - 'false'>
```

En la documentación de la OMG se detalla el estándar de XMI, dando ejemplos de las dos formas tanto el usar tags propios del tipo “ownedElement” con la propiedad “xmi:type” y otros ejemplos con el uso del tag “UML:Class” convalidando cualquiera de las dos formas.

- Importación: Al tomar ejemplos que se desarrollan dentro de los documentos de OMG, no podían ser importados correctamente en ninguna de las herramientas, en ocasiones indicando que faltaba adicionar detalles en la cabecera o algún descriptor del documento, inclusive al adicionarlos tampoco fue posible importarlos. En otros casos informaban que no se podía importar ya que el XMI no estaba formateado correctamente sin detallar a qué se refería exactamente.

En resumen, el formato XMI si bien tiene definiciones de distintos elementos de los metadatos y de los datos, no es lo suficientemente concreto como permitir su tarea principal de poder exportar en una herramienta e importarlo en otra, ya que en muchas ocasiones la importación es parcial o nula.

Tomando una muestra de las herramientas de modelado más difundidas en el mercado se realizó un modelado de ejemplo para comparar los XMI generados y analizar su compatibilidad. Las herramientas analizadas pueden verse en la Tabla 3 así como también la versión de XMI que soportan.

Tabla 3. Soporte de XMI en las herramientas de modelado

Herramienta	XMI
Rational Software Architect	2.1
Enterprise Architect	1.1 ; 2.1
StarUML	1.1
Dia Diagram Editor	No soportado
Modelio	2.1.1 ; 2.2 ; 2.3 ; 2.4.1
Visual Paradigm	1.0 ; 1.2 ; 2.1
Magic Draw Enterprise	2.0

Dentro de los archivos XMI generados por cada herramienta se analizaron los elementos necesarios para importar/exportar un modelo CBMDD, esos elementos son:

- Clases
- Atributos de Clases
- Enumeraciones
- Valores de Enumeraciones
- Estereotipos en Atributos de Clases (Descriptor e Identifier)
- Tipos de Dato
- Componentes
- Estereotipos sobre componentes para determinar su tipo
- Valores Etiquetados

La comparativa de cada uno de los valores de XMI en cada una de las herramientas puede verse en el Anexo A de la presente tesis. El resultado de dicha comparativa muestra la incompatibilidad de los modelos entre sí llegando a la conclusión que no es posible implementar un método genérico para importar y exportar modelos desde la herramienta de CBMDD. Es por ello que a modo de ejemplo, se desarrolló la funcionalidad para una herramienta particular, pudiendo luego extenderse la misma y agregar compatibilidad con otras herramientas. La herramienta elegida fue Enterprise Architect (Sparx Systems, 2015). La elección de dicha herramienta fue debido a que la misma tiene una alta utilización dentro de la industria de software.

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 Figura 20 se muestran las opciones para poder importar o exportar el modelo de datos mediante XMI y la

muestra parte del archivo XMI generado con el modelo de datos de la Figura 20.

Figura 20. Opciones para importar y exportar modelos

```
<?xml version="1.0" encoding="Windows-1252" standalone="yes"?>
<xmi:XMI xmi:version="2.1" xmlns:uml="http://schema.omg.org/spec/UML/2.1" xmlns:xmi="http://schema.omg.org/spec/XMI/2.1"
xmlns:thecustomprofile="http://www.sparxsystems.com/profiles/thecustomprofile/1.0">
  <xmi:Documentation exporter="Enterprise Architect" exporterVersion="6.5" />
  <uml:Model xmi:type="uml:Model" name="EA_Model" visibility="public">
    <packagedElement xmi:type="uml:Package" xmi:id="EAPK_EA4DF52E_871B_497a_B496_1F760BD42505" name="Model" visibility="public">
      <packagedElement xmi:type="uml:Package" xmi:id="EAPK_518D4ABD_9169_46e6_9B7D_1C612CED28E5" name="Class Model" visibility="public">
        <packagedElement name="System" xmi:type="uml:Package" visibility="public" xmi:id="EAPK_4208e47e-20ce-4385-a7e0-5b3bd241c9e4">
          <packagedElement name="Tarea" xmi:type="uml:Class" visibility="public" xmi:id="EAID_bfd9bbb9-975d-4978-baf6-66ea87664110">
            <ownedAttribute name="Titulo" xmi:id="EAID_77e3edc8-ec1f-47b5-9386-9327316eb572" xmi:type="uml:Property" visibility="private"
            isDerived="false">
              <ownedAttribute name="Titulo" xmi:id="EAID_77e3edc8-ec1f-47b5-9386-9327316eb572" xmi:type="uml:Property" visibility=
              "private" isDerived="false" />
            </ownedAttribute>
            <ownedAttribute name="IdTarea" xmi:id="EAID_031435ad-b0c7-45c3-8b4f-cc6eb8aacf9e" xmi:type="uml:Property" visibility="private"
             isDerived="false">
              <ownedAttribute name="IdTarea" xmi:id="EAID_031435ad-b0c7-45c3-8b4f-cc6eb8aacf9e" xmi:type="uml:Property" visibility=
              "private" isDerived="false" />
            </ownedAttribute>
            <ownedAttribute name="Categoria" xmi:id="EAID_cfcc78b4-5635-48c7-8f29-d3f2ff87bcea" xmi:type="uml:Property" visibility=
            "private" isDerived="false">
              <ownedAttribute name="Categoria" xmi:id="EAID_cfcc78b4-5635-48c7-8f29-d3f2ff87bcea" xmi:type="uml:Property" visibility=
              "private" isDerived="false" />
            </ownedAttribute>
```

Figura 21. Archivo XMI generado a partir de un modelo de datos

Al importar el archivo XMI en el Enterprise Architect se puede ver el diagrama tal como fue creado y configurado dentro de CBMDD (ver Figura 22).

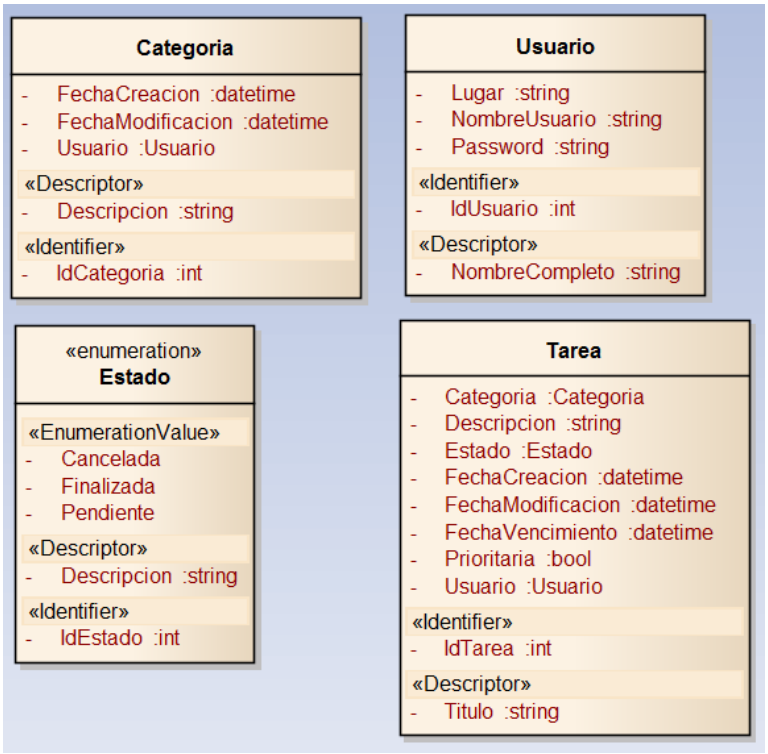

Figura 22. Modelo de datos importado en la herramienta Case mediante un archivo XMI

4.2 Transformaciones

Una transformación es el mapeo de un modelo a otro, en este caso es el mapeo de los modelos de diseño al código fuente de la aplicación.

“Un mapeo se especifica algún lenguaje para describir una transformación de un modelo a otro. Dicha descripción puede estar expresada en lenguaje natural, en un algoritmo en un lenguaje

de programación o en un lenguaje de mapeo de modelos" (OMG, MDA Guide Version 1.0.1, 2003).

Debido a que la información del modelo es almancenada en una base de datos relacional, se opta por desarrollar transformaciones del tipo algorítmicas desarrolladas en el lenguaje de programación C# dentro de la propia herramienta. De forma que la transformación genere nueva información para la propia base de datos o archivos para un programa externo (scripts SQL, código fuente, etc) según el tipo de transformación aplicada.

4.2.1 Del Modelo de Datos al Modelo de Interfaz de Usuario

Esta es la primer transformación de la metodología y tiene como objetivo reducir el trabajo de modelado, ya que partiendo del modelo de datos genera automáticamente un diagrama de componentes completo que luego podrá ser adaptado por el diseñador. Para implementar la transformación en la herramienta se realizó un algoritmo que recorre las tablas del modelo de datos y por cada una va generando los componentes para el modelo de interfaz de usuario. Este proceso se inicia mediante la clase `ClassToComponentes` utilizando el método `GenerateComponents`, que recibe como parámetro el identificador del proyecto del cual se desean generar los componentes. El proceso funciona tanto para generar por primera vez el diagrama de componentes como para actualizarlo luego de alguna modificación en las clases.

En el caso de la primera ejecución, el proceso es directo, y se genera el diagrama de componentes conformado por los componentes `CRUD`, `List` y `Menu` según el procedimiento especificado en la sección 3.3.5.1.

Para el caso de una actualización, la base de datos tiene marcas que permiten identificar el diagrama y los componentes que fueron autogenerados previamente. De forma que sólo se actualizarán dichos componentes, dejando intacto aquellos agregados adicionalmente en la etapa de construcción del modelo de interfaz de usuario.

4.2.2 Del Modelo de Interfaz de Usuario a Código Fuente

Esta es la segunda y última transformación de la metodología. Basada en los modelos y su configuración genera tanto el script de la base de datos como el código fuente en un lenguaje de programación determinado.

4.2.2.1 Script de la base de datos

La generación del script de la base de datos se hace únicamente basándose en el modelo datos siguiendo los pasos especificados en el punto 1 de la sección 3.3.5.2. El script generado además identifica si dicha clase ya había sido exportada a SQL para en caso afirmativo realiza solo un script de actualización para las propiedades modificadas. Esto sirve principalmente para realizar cambios en sistemas ya implementados donde no se puede generar una nueva base de datos sino que se deben preservar los datos actuales y modificar su estructura. Para poder realizar este script diferencial la base de datos cuenta con campos especiales donde está almacenada la información del SQL generada previamente y así poder identificar las diferencias.

4.2.2.2 Generación de código

Al tratarse de un aplicación web móvil se recomienda el uso de la arquitectura Model View Controller MVC (Fowler, 2002) (Deacon, 2013) para poder obtener un código HTML óptimo para los dispositivos móviles.

La implementación de ejemplo genera código para una aplicación Microsoft .NET MVC 4 (Microsoft, ASP.NET MVC 4, 2014) en C#, con una interfaz en HTML5 CON JQuery Mobile (The jQuery Foundation, 2015). Este código incluye todas las clases necesarias (modelos, controladores, acceso a datos y vistas) para ejecutar la aplicación sin necesidad de realizar código adicional.

Las tareas para generar código fuente MVC son:

1. Para cada clase del modelo de datos, se crean las siguientes clases .NET:
    - Un modelo incluyendo las propiedades de la clase.
    - Métodos de acceso a datos para recuperar, crear, actualizar y eliminar registros de la tabla relacionada con la clase del modelo de datos.
    - Un controlador que contendrá los manejadores para las diferentes vistas.
2. Para cada componente en el modelo de interfaz de usuario:
    - Una vista y un método, se añaden en la clase controlador de la entidad configurada como MainEntity en dicho componente. Para aquellos componentes que actualizan datos se crea un método adicional en el controlador con el método http post que permite recibir parámetros y enviar información al modelo para guardar los datos.
    - Si es necesario, se crean clases con vistas para mostrar información particular. Esas clases se agregan dentro una clase especial que contiene todas las vistas ubicada en la carpeta que contiene los modelos.
3. Se toma como base un template de una solución .NET y se actualiza su configuración para:
    - Registrar los nuevos archivos creados en los puntos anteriores.
    - Crear una entrada en el archivo de configuración para guardar el string de conexión con la base de datos.

4.2.3 Clases para facilitar la Transformación a Código Fuente

La herramienta fue desarrollada para facilitar la generación de código en distintas plataformas. Con dicho objetivo para cada componente se ha creado una clase propia (ver Figura 23). Estas clases contienen el método `ProcessAndValidate` que comprueba que el componente esté correctamente configurado. En caso de error retorna un detalle indicando los problemas encontrados y los valores etiquetados que deben ser corregidos.

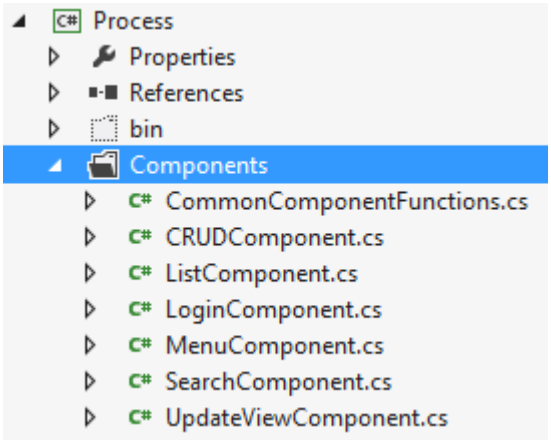

Figura 23. Clases para facilitar la creación de templates para la generación de código

Si la validación es exitosa, ese método completa una serie de propiedades de la clase que deja toda la información pre-procesada para facilitar el desarrollo del algoritmo que realice la generación automática de código. Toda la información textual del modelo es llevada a colecciones fácilmente manejables como Links, Entidades, Propiedades etc. Adicionalmente según el componente también se genera una colección de los Joins SQL necesarios para recuperar los datos relacionados con la entidad principal de dicho componente. A modo de ejemplo se muestran las propiedades disponibles para un componente del tipo `List` en la Figura 24.

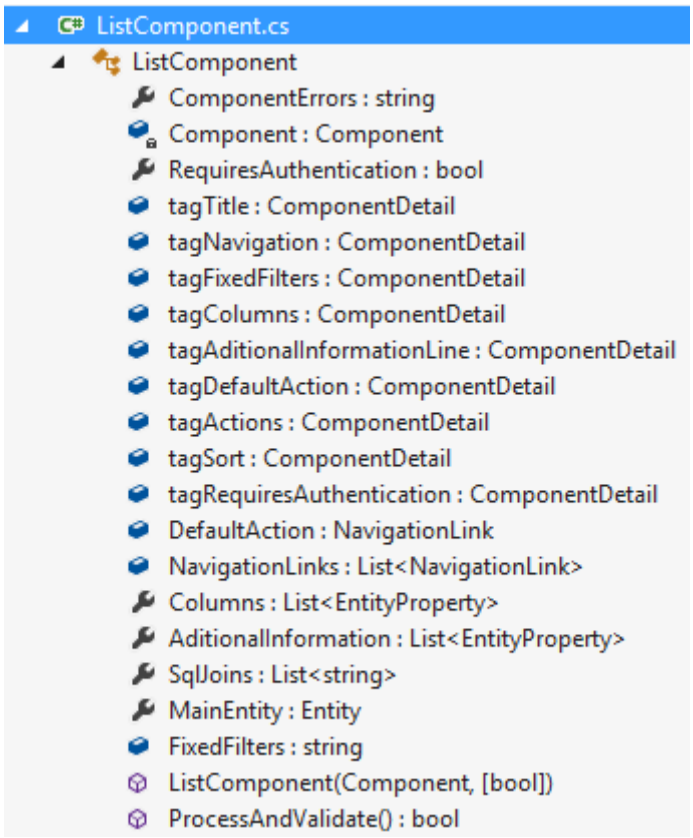

Figura 24. Propiedades disponibles para la generación de código a partir del componente List

Capítulo 5 – Validación

La validación se realizará desde distintos enfoques:

1. Comparando las etapas de la metodología propuesta para CBMDD con la etapas tradicionales de las metodologías de modelado hipermedia.
2. Analizando la validez de la propuesta como un nuevo lenguaje de modelado.
3. Comprobando la efectividad de la metodología, mostrando el modelo realizado y los resultados obtenidos mediante la herramienta.
4. Analizando la aplicabilidad de la metodología a distintos dominios.
5. Evaluando la flexibilidad de la metodología generada.

5.1 Metodología de Modelado

OOHDM (Schwabe & Rossi, An object oriented approach to Web-based applications design, 1998) es uno de los trabajos más reconocidos sobre el modelado hipermedia. En él se establecen cuatro etapas que fueron descriptas en la sección 2.2.3:

1. Diseño Conceptual
2. Diseño Navegacional
3. Diseño de Interfaz Abstracta
4. Implementación

La mayoría de las metodologías de modelado posteriores toman esas etapas como base en su proceso de modelado. CBMDD también incorpora dichas etapas pero unifica el diseño Navegacional y el Diseño de Interfaz Abstracta en un sólo modelo. Las etapas de CBMDD relacionadas con las definidas en OODHM son:

1. Diseño Conceptual: El diseño conceptual, es el modelo de datos de la aplicación que es representado por el diagrama de clases UML.
2. Diseño Navegacional y Diseño de Interfaz Abstracta: El modelo de interfaz de usuario permite establecer los controles que se utilizarán para mostrar la información y además cómo será la navegación entre los distintos componentes del sistema mediante la especificación de los links de navegación y los links de acciones.
3. Implementación: La etapa de implementación no es más ni menos que la segunda transformación propuesta en CBMDD donde partiendo de los modelos se genera en forma automática el código fuente y el script SQL de la aplicación.

5.2 Lenguaje de Modelado

Marco Brambilla en el capítulo 6 de su libro "Model-Driven Software Engineering in Practice" (Brambilla, 2012) establece que: "todo lenguaje de modelado está definido por tres partes principales:

- **Sintaxis Abstracta**: Describe la estructura del lenguaje y la forma en la que las diferentes primitivas pueden combinarse, independientemente de cualquier representación particular o codificación.
- **Sintaxis Concreta**: Describe la representación específica del lenguaje de modelado, cubriendo la codificación y/o cuestiones de apariencia visual. La sintaxis concreta puede ser tanto textual como gráfica.
- **Semántica**: Describe el significado de los elementos definidos en el lenguaje y el significado de las diferentes formas de combinarlos."

Siguiendo estas definiciones se puede comprobar que la metodología presentada en esta tesis cuenta con las 3 partes principales de todo lenguaje de modelado:

- **Sintaxis Abstracta**: Definida mediante el perfil de UML con sus correspondientes restricciones que fue explicado en la sección 3.3.
- **Sintaxis Concreta**: Formada por la especificación de la configuración detallada de cada componente en la sección 3.3.4.4.
- **Semántica**: A lo largo del desarrollo de la sección 3.3 y sus sub-secciones, se ha definido claramente el significado de cada una de las partes de la metodología. Incluyendo el significado de cada valor etiquetado y estereotipo, así como la configuración de cada uno de ellos.

5.3 Efectividad de la Metodología

Para comprobar que la metodología es efectiva para la tarea para cual fue desarrollada se presenta un modelado de un sistema de ejemplo comprobando que siguiendo los pasos establecidos con la herramienta desarrollada se obtiene como resultado una aplicación 100% funcional.

5.3.1 Aplicación a Modelar

Se desea desarrollar una aplicación web móvil donde cada usuario creará una serie de tareas a realizar, pudiendo clasificarlas en distintas categorías y manejando diferentes estados sobre las mismas. Cada usuario administra y visualiza sus propias tareas y categorías. Se asume que los usuarios son dados de alta por un proceso externo fuera de los alcances del sistema móvil.

Los estados de tareas son comunes para todos los usuarios y son:

- Pendiente

- Finalizada
- Cancelada

Por cada tarea la información que se desea registrar es la siguiente:

- Titulo
- Descripción
- Categoría
- Estado
- Fecha de Vencimiento
- Si una tarea es prioritaria o no

Además, se deberá guardar información de auditoria referida a las tareas y categorías de cada usuario. Quedando registrado:

- **FechaCreacion**: Se completa con la fecha y hora del momento de creación del objeto
- **FechaModificacion**: Se completa con la fecha y hora cada vez que se modifica un objeto

La aplicación deberá contar con las siguientes pantallas:

- **Ingreso al Sistema**: Logueo al sistema donde el usuario ingresa al mismo mediante su usuario y password.
- **Menú Principal**: El menú principal contendrá las siguientes opciones:
    - Tareas Prioritarias
    - Tareas Pendientes
    - Agregar Tarea
    - Buscar Tarea
    - Categorías
    - Logout (vuelve a la pantalla de Login)
- **Categorías**: Muestra un listado de las categorías existentes en el sistema. Debe incluir un link para acceder a una pantalla para crear una nueva categoría y otro link para volver al menú principal. El listado de categorías debe solo mostrar la Descripción de la misma y al pulsar en la descripción el usuario será llevado a la pantalla de edición de dicha categoría, donde además podrá eliminarla.
- **Edición de Categoría**: Esta pantalla permite realizar el alta, baja y modificación de una categoría. El usuario solo ingresa la descripción de la misma.
- **Tareas Pendientes**: Esta pantalla deberá mostrar las tareas pendientes del usuario logueado, es decir aquellos objetos de la clase tareas cuya propiedad “usuario”

corresponda al usuario logueado y cuya propiedad “Estado” sea “Pendiente”. Este listado deberá estar ordenado por Fecha de Vencimiento.

La pantalla deberá contener un link para volver al menú principal.

El listado deberá mostrar:

- Titulo
- Categoría
- Fecha de Vencimiento

Al pulsar sobre una fila deberá llevar a la pantalla de edición de dicha tarea.

- **Tareas Prioritarias**: es idéntico al listado de tareas pendientes pero solo deberá mostrar las tareas marcadas como prioritarias y al pulsar sobre una tarea se debe ir a una pantalla para confirmar la misma.
- **Confirmación de Tareas**: esta pantalla muestra información de la tarea y permite confirmarla dando la opción al usuario de modificar los comentarios únicamente.
- **Edición de Tarea**: Esta pantalla permite realizar el alta, baja y modificación de una tarea. Al crear una tarea la misma en forma automática se creará como pendiente, luego en la edición sí, se debe permitir modificar el estado de la misma. Los campos que debe completar el usuario son:
    - Título
    - Descripción
    - Categoría
    - Fecha de vencimiento
    - Estado (solo en la edición)
    - Si es prioritaria o no

  Esta pantalla deberá incluir un link para volver a la pantalla anterior y otro link para volver al menú principal.
- **Búsqueda de Tareas**: Esta pantalla permitirá al usuario mediante diferentes filtros consultar sus tareas vigentes y finalizadas. Se deberá incluir un link para volver al menú principal. Los filtros con que debe contar el usuario para realizar la búsqueda son:
    - Estado: permite elegir un único estado. Debe incluir la opción “Todos”
    - Título: texto libre para buscar una tarea por título.
    - Categoría: permite elegir una única categoría. Debe incluir la opción “Todas”
    - Prioritaria: Filtro del tipo booleano que permite seleccionar, las prioritarias, las no prioritarias o todas.

  Como resultado de la búsqueda se mostrará una grilla con las siguientes columnas:
    - Título de la tarea

- Categoría
- Estado
- Fecha de Vencimiento

Por defecto el listado solo mostrará tareas del usuario logueado.

5.3.2 Modelo de Datos

El primer paso es entonces realizar el modelo de datos mediante el diagrama de clases. Se definen 3 clases y una enumeración. Las clases son:

- Usuario: contendrá la información del usuario para el acceso al sistema.
- Categoría: son las categorías de tareas definidas por cada usuario.
- Tarea: contiene la información de las tareas creadas por cada usuario.

La enumeración corresponde a los Estados de una tarea y puede verse junto con las clases en la Figura 25.

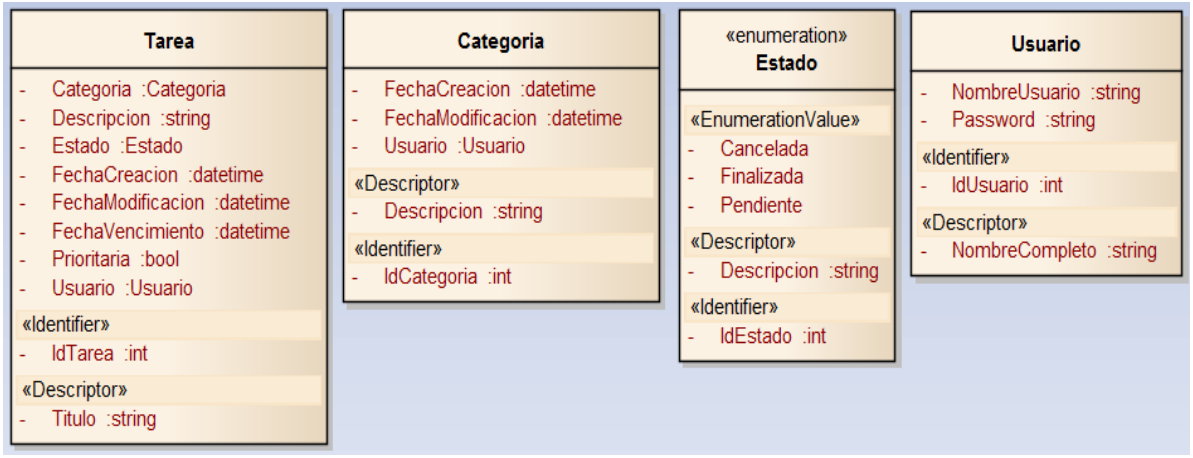

Figura 25. Modelo de datos del sistema de registración de tareas

5.3.3 Modelo de Interfaz de Usuario

Una vez finalizado el modelo de datos se procede a realizar la primera transformación y se obtiene automáticamente el modelo de interfaz de usuario de la Figura 26.

Figura 26. Modelo de Interfaz de Usuario Generado automáticamente a partir del modelo de datos

Sobre ese modelo se realizan las siguientes modificaciones:

1. Se agrega un componente del tipo `Login` como punto de acceso al sistema. Y se lo configura para que una vez autenticado el usuario sea redireccionado al menú. La configuración completa de los valores etiquetados de dicho componente puede verse en la Tabla 4.

Tabla 4. Valores etiquetados del componente de Ingreso al Sistema (tipo `Login`)

Etiqueta	Valor
Id	cpnLogin
Title	Mis Tareas
MainEntity	Usuario
RedirectToComponent	cpnMainMenu
User	NombreUsuario
Password	Password
SelectableUser	False
Navigation	

2. Se eliminan los componentes del listado y edición de usuarios ya que dicha administración se realizará fuera del sistema móvil.

3. Se agrega el componente de listado de tareas prioritarias cuya configuración puede verse en la Tabla 5.

Tabla 5. Valores etiquetados del listado de tareas prioritarias (componente de tipo `List`)

Etiqueta	Valor
Id	cpnLstTareasPrioritarias
Title	Prioritarias
MainEntity	Tarea
Navigation	Link("Home",cpnMainMenu,,0)
RequiresAuthentication	True
FixedFilters	Prioritaria = True AND Tarea.Usuario = LOGGEDUSER
Columns	Tarea.Titulo;
AditionalInformationLine	Categoria,FechaVencimiento

Sort	FechaVencimiento ASC
Actions	
DefaultAction	Link("Confirmar",cpnUdvConfirmarTarea,,)

4. Se personaliza el componente autogenerado del listado de tareas para mostrar las tareas pendientes, para ello se modifica:
    a. El título y Id del componente.
    b. Los filtros por defecto, para mostrar tareas del usuario logueado en estado pendiente.
    c. Se agrega información adicional a mostrar.
    d. Se cambia el orden para que sea por Fecha de Vencimiento en lugar de por título.
    e. Se marca el componente para que requiera autenticación del usuario.
    f. De la barra de navegación se elimina el link para agregar una tarea ya que esa opción se pondrá luego en el menú principal.

    La configuración completa puede verse en la Tabla 6.

Tabla 6. Valores etiquetados del listado de tareas pendientes (componente de tipo `List`)

Etiqueta	Valor
Id	cpnLstTareasPendientes
Title	Pendientes
MainEntity	Tarea
Navigation	Link("Home",cpnMainMenu,,0)
RequiresAuthentication	True
FixedFilters	Tarea.Usuario = LOGGEDUSER AND Estado = PENDIENTE
Columns	Titulo
AditionalInformationLine	Categoria,FechaVencimiento
Sort	Titulo ASC
Actions	
DefaultAction	Link("Editar",cpnCRUDTarea,UD,)

5. Se agrega el componente para buscar tareas cuya configuración es similar a los listados anteriores pero agrega los filtros de búsqueda. La parametrización completa puede verse en la Tabla 7.

Tabla 7. Valores etiquetados de la búsqueda de tareas (componente del tipo `Search`)

Etiqueta	Valor
Id	cpnSchBuscarTareas
Title	Búsqueda
MainEntity	Tarea
Navigation	Link("Home",cpnMainMenu,,0)
RequiresAuthentication	True
FixedFilters	Tarea.Usuario = LOGGEDUSER

Columns	Titulo
AditionalInformationLine	Categoria,FechaVencimiento
Sort	FechaVencimiento ASC
Actions	
DefaultAction	Link("Editar",cpnCRUDTarea,UD,)
SearchFilters	Estado SingleSelection, Titulo FreeText, Categoria SingleSelection, Prioritaria BooleanType
FilterRelatedPropertiesByLoggedUser	True

6. Se agrega el componente para confirmar una tarea mostrando sus datos y permitiendo modificar solo la descripción. Para ello se utiliza un componente del tipo `UpdateView` parametrizado según la Tabla 8.

Tabla 8. Valores etiquetados del componente para confirmar tareas (tipo UpdateView)

Etiqueta	Valor
Id	cpnUdvConfirmarTarea
Title	Confirmar
Navigation	Link("Home",cpnMainMenu,,0)
DisplayProperties	Titulo,Categoria,FechaVencimiento
UpdateProperties	Descripcion
CreateEntity	
DefaultValuesUpdate	Tarea.Estado=Estado.Finalizada
RequiresAuthentication	True
FilterRelatedPropertiesByLoggedUser	True

7. Al componente `menu` se le agrega un link en la barra de navegación con la opción Logout, que volverá a la pantalla de login. Se modifican los links para apuntar a los componentes creados agregando además un link para permitir el alta de tareas. La configuración del componente `menu` puede verse en la Tabla 9.

Tabla 9. Valores etiquetados del Menú Principal (componente del tipo `Menu`)

Etiqueta	Valor
Id	cpnMainMenu
Title	Mis Tareas
Navigation	Link("Logout",cpnLogin,,)
Options	Link("Tareas Prioritarias",cpnLstTareasPrioritarias,,1), Link("Tareas Pendientes",cpnLstTareaPendientes,,2), Link("Agregar Tarea",cpnCRUDTarea,C,3), Link("Buscar Tareas",cpnSchBuscarTareas,,4), Link("Categorías",cpnLstCategoria,,5)
RequiresAuthentication	True

8. En el componente del listado de categorías que fue generado automáticamente se debe poner la marca para indicar que requiere autenticación y configurar para que por defecto

muestre las categorías del usuario logueado. La configuración completa de este componente puede verse en la Tabla 10.

Tabla 10. Valores etiquetados del listado de categorías (componente del tipo List)

Etiqueta	Valor
Id	cpnLstCategoria
Title	Categorias
MainEntity	Categoria
Navigation	Link("Agregar",cpnCRUDCategoria,C,1), Link("Home",cpnMainMenu,,0);)
RequiresAuthentication	True
FixedFilters	Categoria.Usuario = LOGGEDUSER
Columns	Descripcion;
AditionalInformationLine	
Sort	Descripcion ASC
Actions	
DefaultAction	Link("Editar",cpnCRUDCategoria,UD,),

9. Por último se modifican los componentes `CRUD` de categorías y tareas para guardar los datos automáticos para auditoría al momento de creación y modificación y además para registrar el usuario que creó el registro en forma automática. La configuración de dichos componentes, incluyendo los valores por defecto para cada operación, se detallan en la Tabla 11 para el caso de las Tareas y en la Tabla 12 para las categorías.

Tabla 11. Valores etiquetados del componente CRUD de Tareas

Etiqueta	Valor
Id	cpnCRUDTarea
Title	Editar
MainEntity	Tarea
Navigation	Link("Volver",BACK,,), Link("Home",cpnMainMenu,,0);)
DefaultValuesCreate	Tarea.Estado=Estado.Pendiente, Tarea.Usuario=LOGGEDUSER, Tarea.FechaCreacion=NOW
DefaultValuesUpdate	Tarea.FechaModificacion=NOW
SkippedPropertiesCreate	FechaModificacion
SkippedPropertiesUpdate	FechaCreacion, Usuario
CreateEntityOnUpdate	
CreatEntityOnCreate	
OptionalPropertiesCreate	
OptionalPropertiesUpdate	
RequiresAuthentication	True
FilterRelatedPropertiesByLoggedUser	True

Tabla 12. Valores etiquetados del componente CRUD de Categorías

Etiqueta	Valor
Id	cpnCRUDCategoria
Title	Edición de Categoria
MainEntity	Categoria
Navigation	Link("Volver",BACK,,), Link("Menú Principal",cpnMainMenu,,0);)
DefaultValuesCreate	Categoria.FechaCreacion=NOW, Categoria.Usuario=LOGGEDUSER
DefaultValuesUpdate	Categoria.FechaModificacion=NOW
SkippedPropertiesCreate	FechaModificacion
SkippedPropertiesUpdate	Usuario, FechaCreacion
CreateEntityOnUpdate	
CreatEntityOnCreate	
OptionalPropertiesCreate	
OptionalPropertiesUpdate	
RequiresAuthentication	True
FilterRelatedPropertiesByLoggedUser	True

Al configurar los componentes hay que hacerlo teniendo en cuenta el tipo de aplicación que se está diseñando. En el caso de una aplicación web móvil, se debe procurar minimizar la información a buscar en los listados, reduciendo el tamaño de los títulos, etiquetas de los botones, etc. De esta forma la aplicación final será adecuada para ser visualizada en dispositivos con pantalla reducida.

Una vez realizada esta configuración se llega al modelo final de interfaz de usuario cuya vista en el sistema puede verse en la Figura 27.

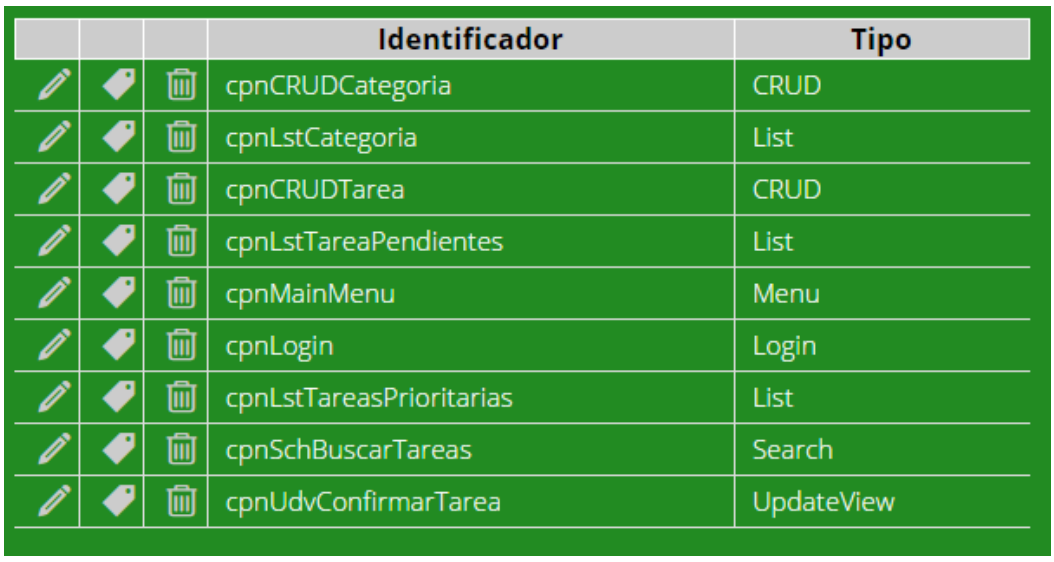

Figura 27.Modelo de Interfaz de Usuario final

Exportando el modelo a XMI e importándolo con el Enterprise Architect podemos ver una vista general de los componentes y sus relaciones en la Figura 28.

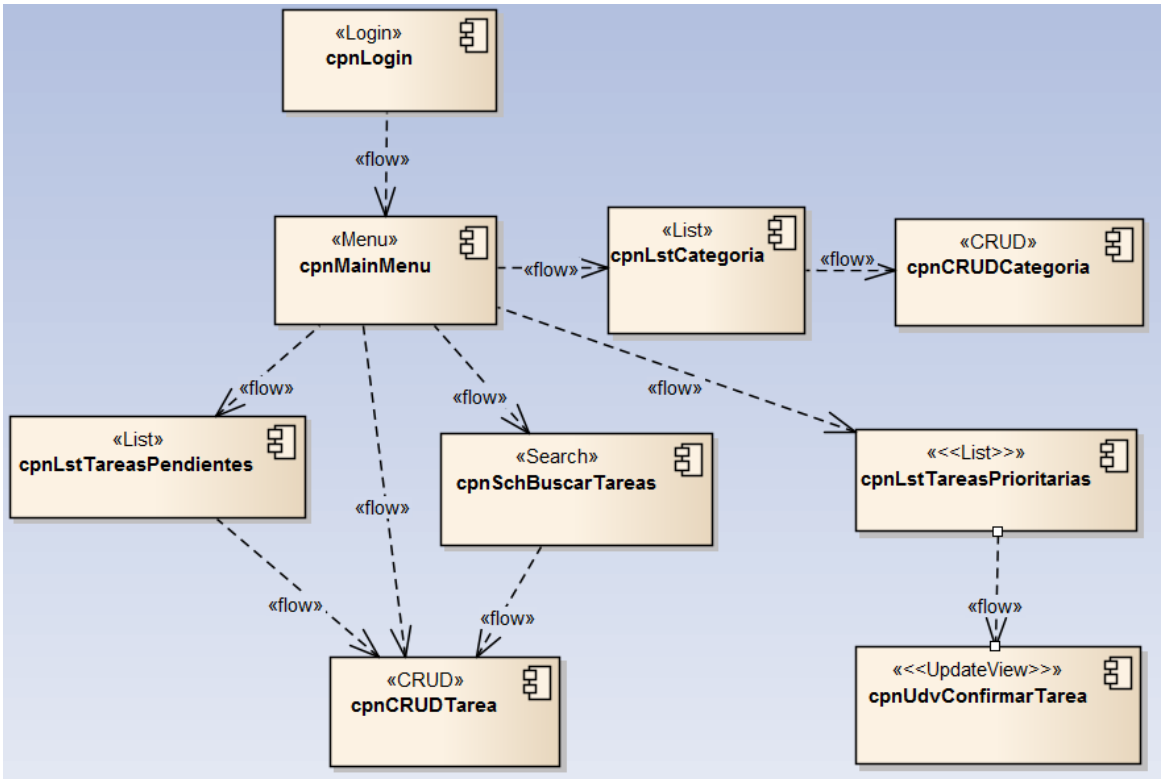

Figura 28. Esquema general de los componentes del sistema

5.3.4 Generación de la Aplicación

Con los dos modelos terminados se puede proceder a generar la aplicación. Para ello se selecciona el template a utilizar y los datos del servidor de base de datos, para poder generar la cadena de conexión correspondiente (ver Figura 29).

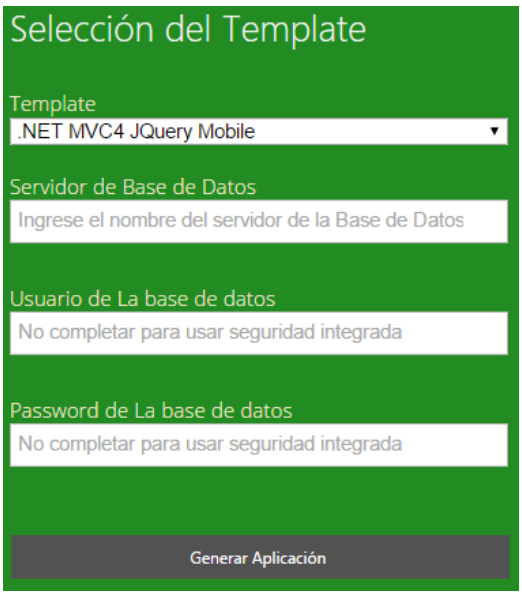

Figura 29. Selección del template para la generación de la aplicación

El proceso genera tanto el script de la base de datos como el código fuente de la misma.

El script SQL generado es:

```
CREATE DATABASE [MisTareas]
GO
USE [MisTareas]
CREATE TABLE Categoria([Descripcion] VARCHAR (100) NOT NULL,[IdCategoria] INTEGER NOT NULL IDENTITY
,[FechaCreacion] DATETIME NULL,[FechaModificacion] DATETIME NULL,[Usuario] INTEGER NOT NULL )
CREATE TABLE Usuario([NombreCompleto] VARCHAR (100) NOT NULL,[IdUsuario] INTEGER NOT NULL IDENTITY
,[NombreUsuario] VARCHAR (100) NULL,[Password] VARCHAR (100) NULL)
CREATE TABLE Estado([Descripcion] VARCHAR (100) NOT NULL,[IdEstado] INTEGER NOT NULL)
INSERT INTO Estado (Descripcion,IdEstado ) values ('Cancelada',1)
INSERT INTO Estado (Descripcion,IdEstado ) values ('Finalizada',2)
INSERT INTO Estado (Descripcion,IdEstado ) values ('Pendiente',3)
CREATE TABLE Tarea([Titulo] VARCHAR (100) NOT NULL,[IdTarea] INTEGER NOT NULL IDENTITY ,[Categoria]
INTEGER NOT NULL ,[Descripcion] VARCHAR (100) NULL,[Estado] INTEGER NOT NULL ,[FechaCreacion]
DATETIME NULL,[FechaModificacion] DATETIME NULL,[FechaVencimiento] DATETIME NULL,[Usuario] INTEGER
NOT NULL ,[Prioritaria] BIT NULL)
ALTER TABLE Tarea ADD CONSTRAINT pk_IdTarea PRIMARY KEY ([IdTarea])
ALTER TABLE Estado ADD CONSTRAINT pk_IdEstado PRIMARY KEY ([IdEstado])
ALTER TABLE Usuario ADD CONSTRAINT pk_IdUsuario PRIMARY KEY ([IdUsuario])
ALTER TABLE Categoria ADD CONSTRAINT pk_IdCategoria PRIMARY KEY ([IdCategoria])
CREATE UNIQUE INDEX IX_CategoriaDescriptor ON [Categoria]([Descripcion])
ALTER TABLE Categoria WITH CHECK ADD CONSTRAINT [FK_Categoria_Usuario] FOREIGN KEY([Usuario])
REFERENCES [dbo].[Usuario] ([IdUsuario])
CREATE UNIQUE INDEX IX_UsuarioDescriptor ON [Usuario]([NombreCompleto])
CREATE UNIQUE INDEX IX_EstadoDescriptor ON [Estado]([Descripcion])
CREATE UNIQUE INDEX IX_TareaDescriptor ON [Tarea]([Titulo])
ALTER TABLE Tarea WITH CHECK ADD CONSTRAINT [FK_Tarea_Categoria] FOREIGN KEY([Categoria])  REFERENCES
[dbo].[Categoria] ([IdCategoria])
ALTER TABLE Tarea WITH CHECK ADD CONSTRAINT [FK_Tarea_Estado] FOREIGN KEY([Estado])  REFERENCES
[dbo].[Estado] ([IdEstado])
ALTER TABLE Tarea WITH CHECK ADD CONSTRAINT [FK_Tarea_Usuario] FOREIGN KEY([Usuario])  REFERENCES
[dbo].[Usuario] ([IdUsuario])
```

Al ejecutar dicho script en un motor de base de datos, como por ejemplo SQL Server, se obtiene la base de datos y las tablas necesarias para la aplicación como puede verse en la Figura 30.

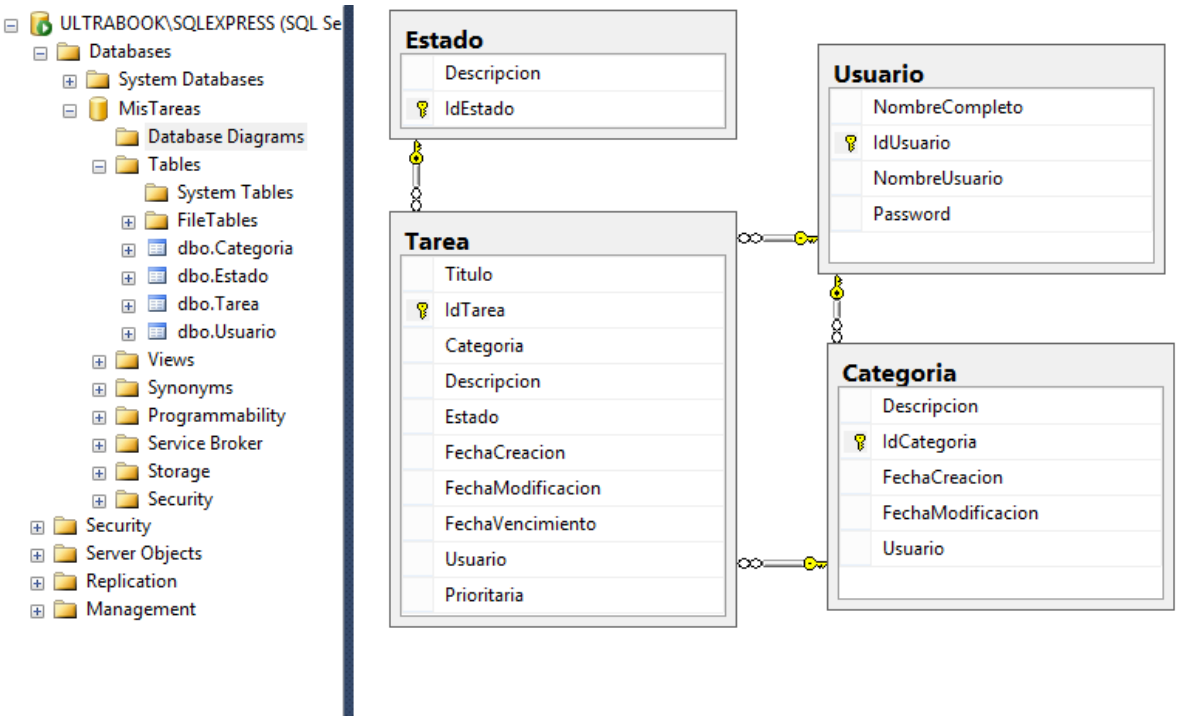

Figura 30. Base de datos generada por el script SQL

La generación de código fuente además de copiar y configurar los archivos del template, crea las clases necesarias para la aplicación MVC tal como fue detallado en la sección 4.2.2.2. La Figura 31 muestra los archivos generados.

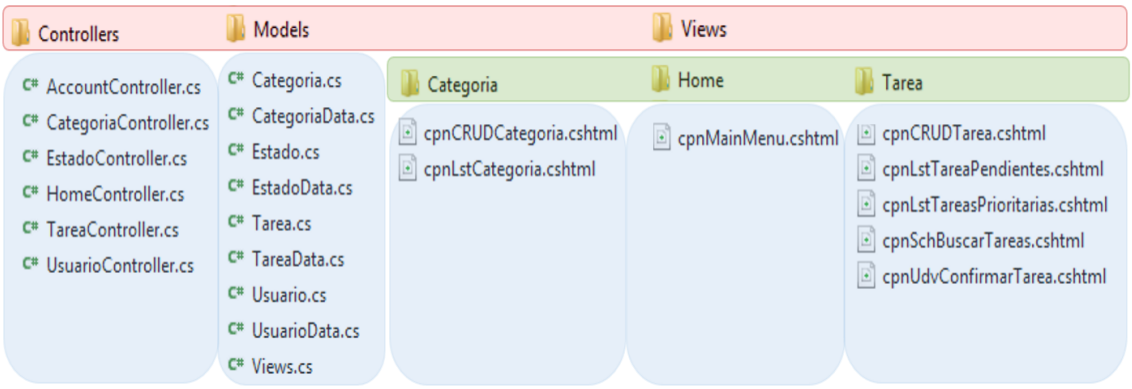

Figura 31. Archivos generados para la aplicación MVC

5.3.5 Ejecución de la Aplicación Generada

Luego de generada la aplicación se obtiene el proyecto que puede abrirse y compilarse con el entorno de programación correspondiente según la transformación realizada. En este caso se generó un proyecto para Microsoft Visual Studio 2012 el cual puede abrirse con el entorno y ejecutarse directamente.

Al ejecutarse la aplicación se mostrara la pantalla de ingreso al sistema. Depende de cómo haya sido configurado el componente Login, el usuario deberá ingresarse manualmente o seleccionarse. El resultado de ambas configuraciones puede verse en la Figura 32.

Figura 32. Pantallas generadas automáticamente a partir de componentes del tipo `Login`

Una vez autenticado el usuario, es redireccionado al menú principal de la aplicación que puede verse en la Figura 33.

Figura 33. Menú principal de la aplicación generado automáticamente

Desde ese menú se accede a las principales opciones del sistema. En esta aplicación se han configurado 3 listados con diferentes configuraciones. En la Figura 34 a la izquierda puede verse el listado de categorías donde se muestra solo la descripción de las mismas. En la barra de navegación hay configurada dos opciones, una para volver al menú principal y la otra para agregar una nueva categoría. Las pantallas del medio y la derecha de la Figura 34 corresponden a los listados de tareas prioritarias y pendientes donde la diferencia son los filtros aplicados para recuperar los datos del listado. Ambos listados muestran varias propiedades de la clase Tarea, en la primera fila de cada ítem se muestra el título de la misma y luego en la fila adicional, la categoría y la fecha de vencimiento.

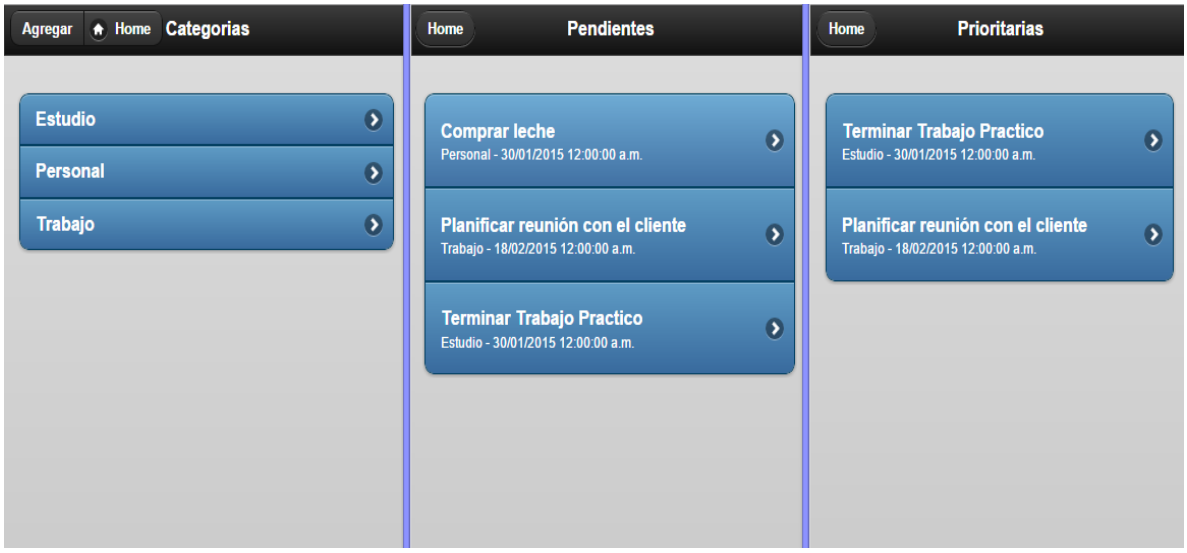

Figura 34. Distintas pantallas generadas a partir de componentes tipo List

Al presionar sobre cada una de las filas de los listados de la Figura 34 se acceden a distintas pantallas que pueden verse en la Figura 35 donde se corresponde la ubicación con la figura anterior. A la izquierda se muestra la pantalla de edición de categorías donde solo se completa la descripción de la misma. En el centro la pantalla de edición de tareas con los distintos campos que pueden ser modificados. Estas dos pantallas corresponden a componentes del tipo CRUD configurados para permitir tanto la edición como la eliminación del registro por eso se muestra tanto el botón para guardar como para eliminar. A la derecha de la Figura 35 se muestra un componente UpdateView configurado para confirmar la tarea donde se visualiza el Título, Categoría y Fecha de Vencimiento y se permite modificar solo la descripción. Al actualizar los datos el estado de la tarea pasará a Finalizada.

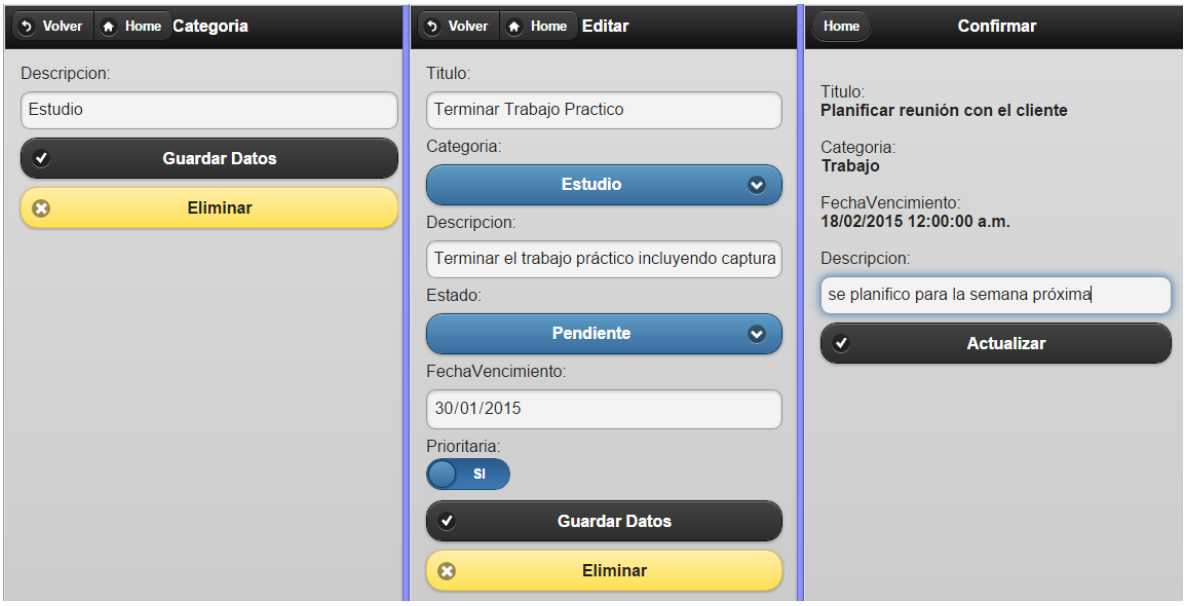

Figura 35. Pantallas de edición generadas a partir de componentes CRUD y UpdateView

Desde el menú principal también se puede acceder a la pantalla de búsqueda de tareas (ver Figura 36). Esta pantalla permite aplicar distintos filtros mostrando como resultado un listado de tareas que cumplan con los filtros seleccionados. Tiene dos filtros de selección simple (Estado y Categoría) un filtro de texto libre (título) y un filtro booleano (Tareas Prioritarias).

Figura 36. Pantalla de Búsqueda generada automáticamente a partir del componente Search

Si desde el menú principal se accede a la opción Agregar Tarea se accede al componente CRUD de tareas (que ya se mostró en la Figura 35 en el centro) pero esta vez recibe el parámetro que sólo habilita el control para crear un registro generando la pantalla de la Figura 37. Puede verse que el estado de la tarea no es solicitado al usuario ya que al crearla la tarea siempre está en estado Pendiente.

Figura 37. Pantalla generada por un componente CRUD configurado para la creación de registros

5.4 Aplicación de la metodología a distintos dominios

Además del ejemplo presentado se ha empleado la metodología en el modelado de una gran cantidad de aplicaciones de distinta complejidad en dominios distintos.

En (Vera, Pons , Giulianelli, & Rodríguez, 2012) se muestra un modelado de un sistema de registración de viajes donde se hace un modelado que incluye un diseño complejo de la base de datos. En esta aplicación se utiliza una tabla separada para registrar los cambios de estado por lo tanto se hace uso del tag `CreateEntity` y además a partir de dicha tabla se recupera información mediante consultas avanzadas utilizando las funciones presentadas en la sección 3.3.4.1. También dicha aplicación hace uso de los links opciones (ver sección 3.3.4.3) para mostrar opciones al usuario solo si cumplen determinadas condiciones sobre los datos.

Al estar basada en controles de interfaz de usuario la metodología no está acotada a un dominio particular. Sino que puede aplicarse en cualquier aplicación que requiera la administración de datos. En el ANEXO B – Modelado de aplicacionesse muestran ejemplos de aplicación de la metodología a dominios diversos, realizando el modelado de aplicaciones con los siguientes objetivos: (1) compra venta de productos, (2) realizar un Log de Viajes y (3) administrar la historia clínica de los pacientes.

La metodología puede entonces aplicarse a distintos dominios y adaptarse para soportar distintos esquemas de bases de datos, aún a aquellos que requieran consultas cruzadas para poder recuperar información relacionada a un objeto.

5.5 Flexibilidad de la metodología

Esta metodología está basada en componentes, tiene un conjunto de componentes predefinidos que servirán para crear una gran gama de aplicaciones. Por supuesto no todas las aplicaciones harán uso de todos los componentes creados, pero es importante analizar que sucede en el caso que se necesiten generar nuevos componentes ó modificar la configuración de componentes existentes.

- Nuevos Componentes: El impacto para agregar un nuevo componente es el siguiente:
    1. Se debe agregar al perfil UML un nuevo estereotipo identificando dicho componente.
    2. Se debe determinar la configuración necesaria para dicho componente y agregar al estereotipo definido valores etiquetados.
    3. Para cada valor etiquetado se debe definir la sintaxis concreta de su configuración que puede coincidir con la forma de configuración ya definida para alguno de los valores etiquetados actuales o en caso de que requiera de una sintaxis diferente, debe detallarse.
    4. Explicar el significado del nuevo tipo de componente y de su configuración definiendo su semántica.
    5. Si es posible y relevante realizar la creación automática de la versión preliminar de dicho componente a partir del modelo de datos.

6. Realizar la transformación a código fuente a partir del nuevo componente.

Siguiendo esta serie de pasos es posible agregar nuevos componentes a la metodología sin variar su funcionamiento general.

- Nuevos parámetros de configuración a uno o más componentes existentes: se debe agregar el valor etiquetado a los estereotipos seleccionados y definir su sintaxis concreta y semántica.

Por lo cual puede decirse que es flexible la metodología y además la herramienta de soporte construida permite dar lugar a estos cambios. Para lo cual deben tomarse las siguientes consideraciones:

- Para agregar un nuevo componente se debe agregar a la tabla ComponentTypes.
- Si hay nuevos valores etiquetados se agregan a la tabla `TaggedValuesTypes` en la cual se define el tipo de control que se utiliza para su configuración, ya contando con una gran cantidad de controles disponibles.
- Por último se vinculan los valores etiquetados con los componentes mediante la tabla `TaggedValuesByComponent`.

Con realizar esta parametrización desde la base de datos, la herramienta ya permitirá crear el nuevo componente y realizar su configuración según los controles seleccionados. Los procesos de transformaciones sí, deben modificarse manualmente.

Capítulo 6 – Conclusiones y Trabajos Futuros

6.1 Conclusiones

El empleo de MDD permite poner el foco de atención en las primeras etapas haciendo que los modelos sean la parte más importante, pudiendo transformarse y evolucionar hasta obtener el código fuente de la aplicación. Es claro que el esfuerzo entonces no está puesto en la programación sino en el diseño. En contrapartida, algunas metodologías de MDD requieren realizar gran cantidad de modelos con inter-relaciones complejas, en donde el esfuerzo final utilizado en las etapas de modelado termina siendo equiparado con el que se requeriría para desarrollar una aplicación de cero de forma tradicional. La presente tesis, plantea la importancia de centrarse en los modelos y como principal contribución teórica se construye una metodología que extiende de forma conservativa a UML, mediante la cual con 2 modelos es posible efectuar transformaciones automáticas de forma transparente al usuario final y obtener el código fuente funcional de una aplicación. Esta metodología se planteó considerando los aspectos mostrados en la Figura 38.

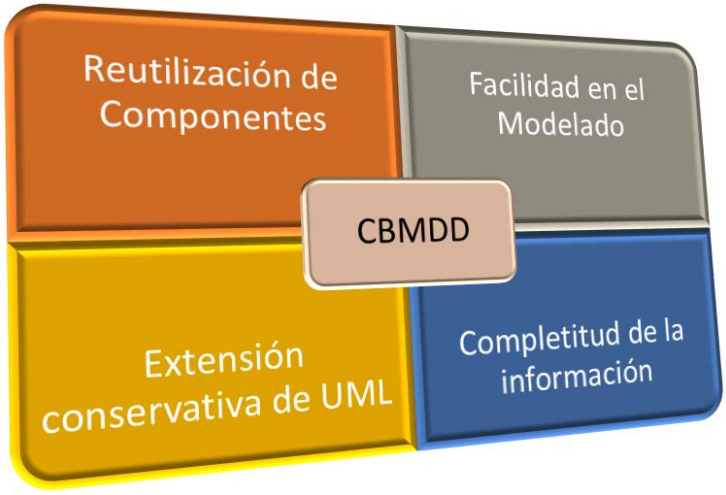

Figura 38. Principales aspectos considerados en definición de la metodología.

Además se desarrolló una herramienta que permite implementar la metodología efectivamente, chequear la trazabilidad entre los modelos generados, importar modelos desde otra herramienta de modelado facilitando la interoperabilidad, efectuar las transformaciones automáticas y generar finalmente como resultado tanto los scripts de la base de datos como el código fuente de la aplicación (ver Figura 39).

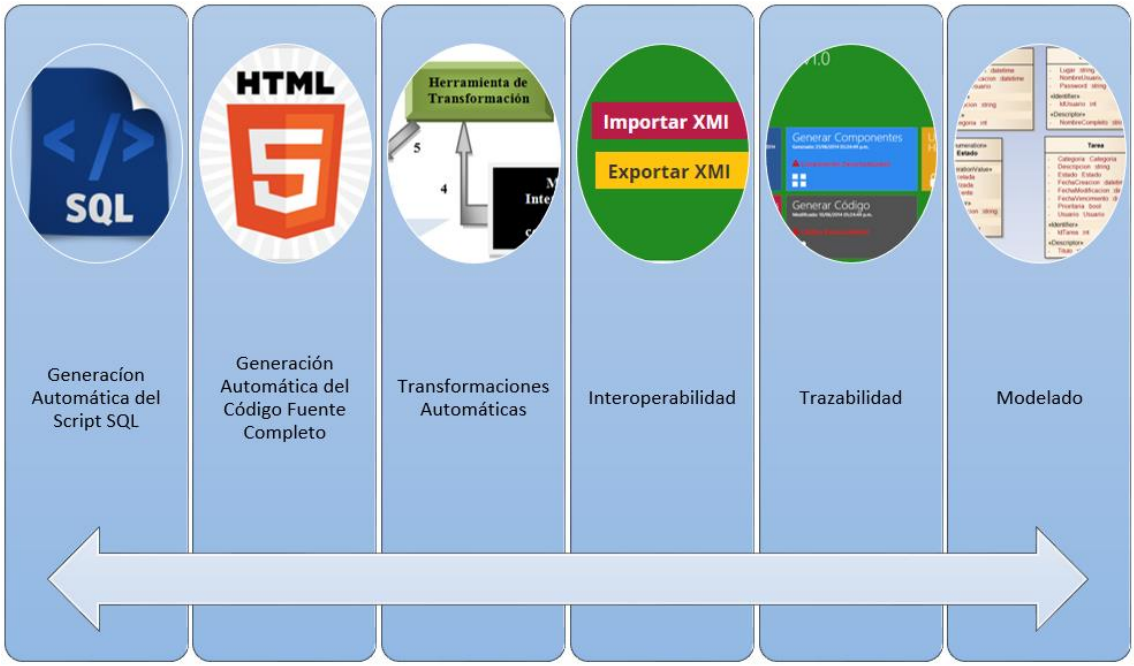

Figura 39. Características principales de la herramienta de soporte a la metodología

La herramienta desarrollada facilita aún más la tarea de modelado ya que evita que el diseñador deba lidiar con la sintaxis concreta de la configuración de los componentes brindando interfaces gráficas de configuración y generación de los modelos. Por otra parte, incorpora los procesos de transformación necesarios en la metodología.

Cabe destacar que la herramienta de soporte fue desarrollada en forma modular permitiendo rápidamente la incorporación de nuevos componentes y nuevos valores etiquetados para permitir la expansión futura de la metodología. También brinda clases para facilitar la tarea de la programación de las transformaciones para generar código fuente incorporando el concepto de templates lo que permite generar con un mismo modelo, código para distintas plataformas.

Si bien la metodología CBMDD ha sido planteada para dispositivos móviles es posible adaptarla a otras plataformas. En el caso de que la interfaz de usuario que se desee modelar tenga una distribución de pantalla más compleja es posible agregar nuevos componentes así como también agregar parámetros de configuración a los componentes existentes según las necesidades del caso.

6.2 Trabajos Futuros

Como trabajos futuros se planea:

- La creación de más templates para generar interfaces para distintas plataformas y distintos lenguajes.
- Incorporar opciones de interoperabilidad con más herramientas de modelado.
- Hacer más poderosos los templates de creación de código fuente incorporando parámetros a los mismos para generar interfaces con distintas opciones sin necesidad

de crear un nuevo template. Por ejemplo podría seleccionar si la navegación primaria es persistente (siempre fija) o transitoria (se despliega un menú con una opción) en la pantalla. O incluso permitir seleccionar imágenes para el menú principal sea icónico en lugar de textual (patrón Springboard).

- Incorporación de nuevos controles o nuevas configuraciones para hacer más poderoso el modelo.

6.3 Publicaciones Efectuadas

En esta sección se listan las distintas publicaciones realizadas referidas al trabajo de la presente tesis en orden cronológico. En ellas puede verse la evolución del trabajo en sus distintas etapas con cambios y mejoras hasta llegar a la versión final. Incluso el nombre de la metodología fue cambiado de CBHDM a CBMDD por reflejar este último mejor la propuesta realizada, es por eso que muchas de las publicaciones fueron realizadas con el primer nombre asignado a la metodología. Por cada una de ellas además se incorpora una breve explicación para determinar la evolución de la metodología a lo largo de las publicaciones. Las última publicación se encuentran en edición debido a los tiempos de publicación del Journal pero ya ha sido aprobada y enviada su versión final.

1. VII Congreso Colombiano de Computación (7CCC) – 2012
   Título: User Interface and Navigation Modeling Methodology for Mobile Hypermedia Systems
   ISBN: 978-1-4673-1475-6
   Autores: Pablo Vera, Daniel Giulianelli, Rocío Rodríguez, Claudia Pons
   Lugar: Medellín, Colombia
   Comentarios: Esta es la primera publicación, donde se presenta la idea general de la metodología, mostrando sus componentes, valores etiquetados y un ejemplo de aplicación.

2. XV Workshop de Investigadores en Ciencias de la Computación (WICC) - 2012
   Título: Utilizando el Enfoque MDA para la Construcción de Aplicaciones Web Móviles Centradas en los Datos
   ISBN: 978-950-766-082-5
   Autores: Pablo Vera, Claudia Pons, Daniel Giulianelli, Rocío Rodríguez
   Lugar: Universidad Nacional de Misiones, Posadas, Misiones, Argentina
   Comentarios: En este workshop se presenta en forma general el comienzo de la línea de investigación donde inicialmente se había planteado la metodología con tres modelos en lugar de dos como la versión final.

3. XVII Congreso Argentino de Ciencias de la Computación (CACIC) - 2012
   Título: MDA based Hypermedia Modeling Methodology using reusable components
   ISBN: 978-987-1648-34-4
   Autores: Pablo Vera, Claudia Pons, Daniel Giulianelli, Rocío Rodríguez
   Lugar: Universidad Nacional del Sur, Bahía Blanca, Buenos Aires, Argentina

Comentarios: En este trabajo se detallan detenidamente los pasos para las distintas transformaciones de código. Se renombra la propiedad `Filters` del component List para unificar criterios con el componente `Search`. Se incorpora el componente `UpdateView`. También se propone un primer enfoque para crear un front end no móvil utilizando un template separado.

4. XV Workshop de Investigadores en Ciencias de la Computación (WICC) - 2013
   Título: Metodología de Modelado de Aplicaciones Web Móviles Basada en Componentes
   ISBN: 978-987-28179-6-1
   Autores: Pablo Vera, Claudia Pons, Carina González, Daniel Giulianelli, Rocío Rodríguez
   Lugar: Universidad Autónoma de Entre Ríos (UADER), Paraná, Entre Ríos, Argentina
   Comentarios: Se avanza con el detalle de la línea de investigación y se incorpora la idea del desarrollo de una herramienta de soporte de modelado.

5. XLII Jornadas Argentinas de Informática (42 JAIIO) – 2013
   Título: Metodología de Modelado de Aplicaciones Web Móviles Basada en Componentes de Interfaz de Usuario
   ISSN: 1850-2776
   Autores: Pablo Vera, Claudia Pons, Carina González González, Daniel Giulianelli, Rocío Rodríguez
   Lugar: Facultad de Matemática, Astronomía y Física de la Universidad Nacional de Córdoba, Córdoba Capital, Córdoba, Argentina.
   Comentarios: se agrega configuración en los componentes para poder generar menús contextuales al poder definir más de una acción sobre un ítem de un listado. Al mismo tiempo se agrega el concepto de acción por defecto en el caso de que sólo se disponga de una opción y se presenta una primera versión del perfil de UML de la metodología.

6. XIX Congreso Argentino de Ciencias de la Computación (CACIC) - 2013
   Título: Modeling Complex Mobile Web Applications from UI Components – Adding Different Roles and complex Database Design
   ISBN: 978-987-23963-1-2
   Autores: Pablo Vera, Claudia Pons, Carina González González, Daniel Giulianelli, Rocío Rodríguez
   Lugar: Universidad CAECE, Mar del Plata, Buenos Aires, Argentina.
   Comentarios: En este trabajo se muestra cómo utilizar el componente `UpdateView` junto con las funciones de recuperación de datos y links opcionales para modelar aplicaciones con diseño de base de datos complejo. Además se incorpora un nuevo parámetro a la función de links para permitir generar opciones de menú dependiente del rol del usuario logueado (que luego será unificado en el campo condición de la función de links).

7. Third International Conference on Software and Emerging Technologies for Education, Culture, Entertainment, and Commerce (SETECEC): New Directions in Multimedia Mobile Computing, Social Networks, Human-Computer Interaction and Communicability - 2014
   Título: Tool for developing Mobile Web Application from UI Models – Based on CBHDM Methodology
   ISBN: 978-88-96-471-27-2
   Autores: Pablo Vera, Claudia Pons, Carina González González, Rocío Rodríguez, Daniel Giulianelli
   Lugar: Venecia, Italia
   Comentarios: Este trabajo se centra en la herramienta, detallando como permite aplicar la metodología, en cuanto a la construcción de los distintos modelos y sus transformaciones. Se realizó una demo en vivo del funcionamiento, mostrando la rapidez con la cual se puede desarrollar un modelado y como la herramienta da soporte al proceso de configuración.

8. XVI Workshop de Investigadores en Ciencias de la Computación (WICC) - 2014
   Título: Generación Automática de Aplicaciones Web Móviles Mediante Componentes Configurables
   ISBN: 978-950-34-1084-4
   Autores: Pablo Vera, Claudia Pons, Carina González, Rocío Rodríguez, Daniel Giulianelli
   Lugar: Universidad Nacional de Tierra del Fuego, Ushuaia, Tierra del Fuego, Argentina
   Comentarios: En este trabajo se muestra el desarrollo de la herramienta de soporte de modelado para metodología junto con los primeros resultados obtenidos sobre la generación automática de código.

9. XX Congreso Argentino de Ciencias de la Computación (CACIC) - 2014
   Título: Automatic Creation of Mobile Web Applications from Design Models
   ISBN: 978-987-3806-05-6
   Autores: Pablo Martín Vera, Claudia Pons, Carina González González, Rocío Andrea Rodríguez, Daniel Alberto Giulianelli
   Lugar: Universidad Nacional de La Matanza, San Justo, Buenos Aires, Argentina
   Comentarios: Ya con la herramienta finalizada, este trabajo se centra en los resultados, mostrando para cada componente como son las pantallas finales que se generan automáticamente, mostrando las variaciones al cambiar la configuración de las mismas. Este trabajo fue galardonado como “Mejor Exposición” dentro del Workshop de Ingeniería de Software.

10. International Journal of Information Technologies and Systems Approach (IJITSA) - Volúmen 8, Número 2, pp 80 – 100. Special Issue on HCI - 2015
    Título: Component Based Model Driven Development – An Approach for Creating Mobile Web Applications from Design Models
    ISSN: 1935-570X
    Autor: Pablo Martín Vera

Comentarios: En este Journal se cambia el nombre de la metodología a CBMDD además de explicar la ventaja de la utilización de componentes en MDD. Se muestra también el perfil de UML actualizado explicando cada una de las extensiones realizadas. Dada la extensión del artículo también permitió presentar en detalle cada una de las secciones de la metodología.

11. Revista Colombiana de Computación (RCC) – 2015 (en edición)
    Título: La interfaz de usuario como punto de partida para la creación automática de aplicaciones móviles – Un enfoque basado en MDD
    Autores: Pablo M. Vera, Claudia Pons, Carina González González, Rocío A. Rodríguez
    Comentarios: En este trabajo se hace hincapié en la utilización de componentes de interfaz de usuario para el modelado justificando la utilización de componentes y su configuración. Además se presenta la versión final del perfil UML tal como se muestra en la presente tesis.

Acrónimos

A

ANSI	American National Standards Institute
ASP	Active Server Pages

C

CBMDD	Component Based Model Driven Development
CRUD	Create, Read, Update, Delete

D

DSL	Domain Specific Language

G

GPS	Global Positioning System

H

HTML	HyperText Markup Language
HDM	Hypertext Design Model

I

IEEE	Institute of Electrical and Electronics Engineers
IFML	Interaction Flow Modeling Language

J

JDBC	Java Database Connectivity

M

MDA	Model Driven Architecture
MDD	Model Driven Development
MDE	Model Driven Engineering
MOF	Meta Object Facility
MSE	Model Driven Software Engineering
MVC	Model View Controller

O

OMG	Object Management Group
OOHDM	Object Oriented Hipermedia Design Method
OQL	Object Query language

P

PIM	Platform Independent Model
PSM	Platform Specific Model
PDA	Personal Digital Assistant

R

RMM	Relationship Management Methodology
RMDM	Relationship Management Data Model

S

SMS	Short Message Service
SQL	Structured Query Language

U

UI	User Interface
UML	Unified Modeling Language
URL	Uniform Resource Locator
UWE	UML Based Web Engineering

W

W3C	World Wide Web Consortium
WEBML	Web Modeling Language
WSDM	Web Site Design Method

X

XMI	XML Metadata Interchange
XML	Extensible Markup Language

Referencias

Adler, R. M. (1995). Emerging standards for component software. *Computer, 28*(3), 68-77.

Alonso Diego, P. J.-C. (2012). Generación Automática de Software para Sistemas de Tiempo Real: Un Enfoque basado en Componentes, Modelos y Frameworks. *Revista Iberoamericana de Automática e Informática Industrial RIAI, 9*(2), 170-181.

Atzeni Paolo, M. G. (1998). Design and maintenance of data-intensive web sites. In *Advances in Database Technology — EDBT'98* (pp. 436-450). Springer Berlin Heidelberg.

Balagtas-Fernandez Florence. (2008). Model-Driven Development of Mobile Applications. *23rd IEEE/ACM International Conference on Automated Software Engineering* (pp. 509 - 512). L'Aquila: IEEE.

Baumeister Hubert, K. N. (1999). Towards a UML Extension for Hypermedia Design. *Lecture Notes in Computer Science*, *1723*, pp. 614-629.

Bencomo Nelly, G. P. (2008). 4. Genie: Supporting the Model Driven Development of Reflective, Component-based Adaptive Systems. *Proceedings of the 30th international conference on Software engineering* (pp. 811-814). Nueva York, Estados Unidos: ACM New York.

Booch Grady, B. A. (2014). An MDA Manifesto. *MDA Journal*.

Brambilla Marco, B. S. (2014). Fifteen Years of Industrial Model-Driven Development in Software Front-End: from WebML to WebRatio and IFML. *Novática*(228), 36.

Brambilla Marco, F. P. (2014). *Interaction Flow Modeling Language: Model-Driven UI Engineering of Web and Mobile Apps with IFML.* Morgan Kaufmann - Elsevier.

Brambilla, M. C. (2012). *Model-driven software engineering in practice* (Vol. 1). Synthesis Lectures on Software Engineering.

Ceri, S., Fraternali, P., & Bongio, A. (2000). Web Modeling Language (WebML): a modeling language for designing Web sites. *Computer Networks, 33*(1-6), 137–157.

De Troyer O.M.F., L. (1998). WSDM: A User Centered Design Method for Web Sites. *Proceedings of the 7th International World Wide Web* (pp. 85 - 94). Elsevier.

Deacon, J. (2013). *Model-view-controller (mvc) architecture.* Retrieved 12 7, 2014, from http://www.jdl.co.uk/briefings/MVC.pdf

Fowler Martin, P. R. (2010). *Domain Specific Languages .* Addison-Wesley Educational Publishers Inc.

Fowler, M. (2002). *Patterns of Enterprise Application Architecture.* Boston, MA, USA: Addison-Wesley Longman Publishing Co.

France Robert, R. B. (2005). Domain specific modeling. *Software & Systems Modeling, 4*(1), 1-3.

Fuentes, L., & Vallecillo, A. (2004). Una Introducción a los Perfiles UML. *Revista Novatica– Asociación de Técnicos de Informática-España*.

Garzotto, F., Paolini, P., & Schwabe, D. (1993). HDM—a model-based approach to hypertext application design. *CM Transactions on Information Systems (TOIS), 11*(1), 1-26.

Hailpern B., T. P. (2006). Model-driven development: The good, the bad, and the ugly. *IBM Systems Journal, 45*(3), 451 - 461.

Heineman, G. T., & Councill, W. T. (2001). *Component-Based Software Engineering: Putting the Pieces Together.* Addison-Wesley.

IEEE. (2010, 03 11). *Standards Glossary*. Retrieved 01 21, 2015, from http://www.ieee.org/education_careers/education/standards/standards_glossary.html

Isakowitz, T., Stohr, E., & Balasubramanian, P. (1995). RMM: a methodology for structured hypermedia design. *Communications of the ACM*, 34-44.

Johannes, J. (2011). Component Based Model-Driven Software Development. *PhD Thesis*. Technischen Universit¨at Dresden.

Keller, R. K., & Reinhard, S. (1998). Design components: toward software composition at the design level. *Proceedings of the 20th international conference on Software engineering.* IEEE Computer Society.

Kleppe, A., Warmer, J., & Wim, B. (2003). *MDA explained: the model driven architecture: practice and promise.* Addison-Wesley Professional.

Koch Nora, A. K. (2002). The Expressive Power of UML-based Web Engineering. *Second International Workshop on Web-oriented Software Technology (IWWOST02)*, *16.*

Koch, N., Knapp, A., Zhang, G., & Baumeister, H. (2008). Uml-Based Web Engineering. In *Web Engineering: Modelling and Implementing Web Applications* (pp. 157-191). Springer London.

Krug, S. (2006). *No me hagas Pensar: Una aproximación a la usabilidad en la web.* Pearson Prentice Hall.

Kulkarni Vinay, R. S. (2008). An abstraction for reusable MDD components: model-based generation of model-based code generators. *Proceedings of the 7th international conference on Generative programming and component engineering* (pp. 181-184). Nueva York, Estados Unidos: ACM New York.

Landow, G. (1995). *Hipertexto: la convergencia de la teoría crítica contemporánea y la tecnología.* Barcelona: Paidós.

Liu, Z., Morisset, C., & Stolz, V. (2010). rCOS: Theory and Tool for Component-Based Model Driven Development. In *Fundamentals of Software Engineering* (Vol. 5961, pp. 62-80). Springer Berlin Heidelberg.

Mernik Marjan, H. J. (2005, 12). When and how to develop domain-specific languages. *ACM Computing Surveys (CSUR), 37*(4), 316-344.

Microsoft. (n.d.). Retrieved 01 05, 2015, from ASP.NET Web Forms: http://www.asp.net/web-forms

Microsoft. (2014). *ASP.NET MVC 4*. Retrieved 12 7, 2014, from http://www.asp.net/mvc/mvc4

Nielsen, J., & Budio, R. (2012). *Mobile Usability.* New Riders.

Odutola, K., & van der Wulp, M. (2010). *ArgoUML Quick Guide*. Retrieved 01 15, 2015, from http://argouml-stats.tigris.org/documentation/quickguide-0.34/

OMG. (2003). *MDA Guide Version 1.0.1.* Retrieved 11 04, 2014, from http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf

OMG. (2013, 03). *Interaction Flow Modeling Language.* Retrieved 02 21, 2015, from http://www.omg.org/spec/IFML/1.0/Beta1/

OMG. (2014, 04). *XML Metadata Interchange (XMI), Version 2.4.2*. Retrieved 01 19, 2015, from http://www.omg.org/spec/XMI/2.4.2/

OMG. (n.d.). *OMG's MetaObject Facility*. Retrieved 03 10, 2015, from http://www.omg.org/mof/

OMG. (n.d.). *Unified Modeling Language - Resource Page*. Retrieved 02 24, 2015

Oracle. (n.d.). *JavaServer Pages Technology.* Retrieved 02 12, 2015, from http://www.oracle.com/technetwork/java/javaee/jsp/index.html

Papajorgji, P., Beck, H. W., & Braga, J. L. (2004). n architecture for developing service-oriented and component-based environmental models. *Ecological Modelling, 179*(1), 61-76.

PhungKhac, A., Beugnard, A., Gilliot, J. M., & Segarra, M. T. (2008). Model Driven Development of Component based Adaptive Distributed Applications. *Proceeding of the 23rd ACM Symposium on Applied Computing (SAC'2008), track on Dependable and Adaptive Distributed Systems (DADS.* Fortaleza, Ceara, Brazil: ACM.

Piero Fraternali, P. P. (1998). A conceptual model and a tool environment for developing more scalable, dynamic, and customizable Web applications. In *Advances in Database Technology — EDBT'98* (Vol. 1377, pp. 419-435). Springer.

Puerta Angel, M. M. (2005). The UI pilot: a model-based tool to guide early interface design. *Proceedings of the 10th international conference on Intelligent user interface* (pp. 215-222). Nueva York, Estados Unidos: ACM New York.

Radeke Frank, F. P. (2006). PIM Tool: Support for Pattern-Driven and Model-Based UI Development. *Task Models and Diagrams for Users Interface Design - 5th International Workshop, TAMODIA* (pp. 82-96). Hasselt, Belgium: Springer Berlin Heidelberg.

Rossi, G., Schwabe, D., Olsina, L., & Pastor, O. (2008). *Web Engineering: Modelling and Implementing Web Applications.*

Schwabe, D., & Rossi, G. (1995). Building hypermedia applications as navigational views of Information Models. *Proceedings of the 28th Annual Hawaii International Conference on System Science.*

Schwabe, D., & Rossi, G. (1998). An object oriented approach to Web-based applications design. *Theory and Practice of Object Systems - Special issue objects, databases, and the WWW*, 207-225.

Sparx Systems. (2015, 01 19). *Enterprise Architect - Herramienta de diseño UML*. Retrieved from http://www.sparxsystems.com.ar/products/ea.html

Steimann, F., & Kühne, T. (2005, 12). Coding for the Code. (ACM, Ed.) *Queue - Managing Megaservice, 3*(10), 44-51.

The jQuery Foundation. (2015). *jQuery Mobile*. Retrieved 02 05, 2015, from http://jquerymobile.com/

Thompson Chris, W. J. (2009). Optimizing Mobile Application Performance with Model–Driven Engineering. *Proceedings of 7th IFIP WG 10.2 International Workshop* (pp. 36-46). Newport Beach, CA, USA: Springer Berlin Heidelberg.

UC San Diego Computer Science and Engineering. (n.d.). *OQL - Object Query Language*. Retrieved 1 10, 2015, from http://cseweb.ucsd.edu/classes/wi00/cse132a/oql.htm

Vera, P., Pons , C., Giulianelli, D., & Rodríguez, R. (2012). MDA based Hypermedia Modeling Methodology using reusable components. *XVIII Congreso Argentino de Ciencias de la Computación.*

W3C. (2009, 07 29). *Mobile Web Best Practices 1.0*. Retrieved 01 05, 2015, from http://www.w3.org/TR/mobile-bp/

W3C. (2010, 12 14). *Mobile Web Application Best Practices.* Retrieved 09 17, 2014, from http://www.w3.org/TR/mwabp/

Weiss K., O. E. (2003). Reusable Specification Components for Model-Driven Development. *Proceedings of the International Conference on System Engineering, INCOSE .*

Witt Hendrik, N. T. (2007). The WUI-Toolkit: A Model-Driven UI Development Framework for Wearable User Interfaces. *27th International Conference on Distributed Computing Systems Workshops* (p. 43). Toronto: IEEE.

ANEXO A - Comparativa de XMI entre distintas herramientas

En este anexo se presenta en forma resumida, a través de un conjunto de tablas, la comparativa entre los XMI resultantes de exportar un mismo modelo en distintas herramientas. Se analizan los elementos de interés para la metodología CBMDD propuesta en esta tesis.

Tabla 13. Comparativa de XMI para las Clases

Clase				
Herramienta	**Contenedor**	**Tipo XMI**	**Nombre**	**Arbol XMI**
Enterprise Architect	packagedElement	uml:Class	name	<xmi:XMI> <uml:Model xmi:type="uml:Model"> <packagedElement xmi:type="uml:Package"> <packagedElement xmi:type="uml:Package"> <packagedElement xmi:type="uml:Class" name="NOMBREdeCLASE">
Modelio	packagedElement	uml:Class	name	<uml:Model> <packagedElement xmi:type="uml:Class" name="NOMBREdeCLASE”>
Rational Software Architect	PackagedElement	uml:Class	name	<uml:Package> <packagedElement xmi:type="uml:Class" name="NOMBREdeCLASE">
Visual Paradigm	ownedMember	uml:Class	name	<xmi:XMI> <uml:Model name="NOMBREdelMODELO"> <ownedMember name="NOMBREdeCLASE" xmi:type="uml:Class">
MagicDraw Enterprise	nestedClassifier	uml:Class	name	<xmi:XMI> <uml:Model name="NOMBREdelMODELO"> <packagedElement> <nestedClassifier xmi:type="uml:Class" name="NOMBREdeCLASE">

Tabla 14. Comparativa de XMI para atributos de una Clase

Atributo de Clase				
Herramienta	**Contenedor**	**Tipo XMI**	**Nombre**	**Arbol XMI**
Enterprise Architect	ownedAttribute	uml:Property	name	<xmi:XMI> <uml:Model xmi:type="uml:Model"> <packagedElement xmi:type="uml:Package"> <packagedElement xmi:type="uml:Package"> <packagedElement xmi:type="uml:Class"> <ownedAttribute xmi:type="uml:Property"name="NOMBREdeATRIBUTO">
Modelio	packagedElement	uml:Property	name	<uml:Model> <packagedElement xmi:type="uml:Class" name="NOMBREdeCLASE”> <ownedAttribute xmi:type="uml:Property" name="NOMBREdeATRIBUTO">
Rational Software Architect	ownedAttribute	uml:Property	name	<uml:Package> <packagedElement xmi:type="uml:Class" name="NOMBREdeCLASE"> <ownedAttribute xmi:type="uml:Property" name="NOMBREdeATRIBUTO">
Visual Paradigm	ownedAttribute	uml:Property	name	<xmi:XMI> <uml:Model name="NOMBREdelMODELO"> <ownedMember name="NOMBREdeCLASE" xmi:type="uml:Class"> <ownedAttribute name="NOMBREdelATRIBUTO" xmi:type="uml:Property">
MagicDraw Enterprise	ownedAttribute	uml:Property	name	<xmi:XMI> <uml:Model name="NOMBREdelMODELO"> <packagedElement> <nestedClassifier xmi:type="uml:Class" name="NOMBREdeCLASE"> <ownedAttribute name="NOMBREdeATRIBUTO">

Tabla 15. Comparativa de XMI para las Enumeraciones

Enumeración				
Herramienta	**Contenedor**	**Tipo XMI**	**Nombre**	**Arbol XMI**
Enterprise Architect	packagedElement	uml:Enumeration	name	<xmi:XMI> <uml:Model xmi:type="uml:Model"> <packagedElement xmi:type="uml:Package"> <packagedElement xmi:type="uml:Package"> <packagedElement xmi:type="uml:Enumeration" name="NOMBREdeENUMERACION">
Modelio	packagedElement	uml:Enumeration	name	<uml:Model> <packagedElement xmi:type="uml:Enumeration" name="NOMBRE">
Rational Software Architect	packagedElement	uml:Enumeration	name	<uml:Package> <packagedElement xmi:type="uml:Enumeration" name="NOMBRE">
Visual Paradigm	ownedMember	uml:Enumeration	name	<xmi:XMI> <uml:Model> <ownedMember name="NombreDeEnumeracion" xmi:type="uml:Enumeration">
MagicDraw Enterprise	nestedClassifier	uml:Enumeration	name	<xmi:XMI> <uml:Model name="NOMBREdelMODELO"> <packagedElement> <nestedClassifier xmi:type="uml:Enumeration" name="NOMBREdeENUMERACION">

Tabla 16. Comparativa de XMI para los valores de las Enumeraciones

Valor de Enumeración				
Herramienta	**Contenedor**	**Tipo XMI**	**Nombre**	**Arbol XMI**
Enterprise Architect	ownedAttribute	uml:Property	name	<xmi:XMI> <uml:Model xmi:type="uml:Model"> <packagedElement xmi:type="uml:Package"> <packagedElement xmi:type="uml:Package"> <packagedElement xmi:type="uml:Enumeration"> <ownedAttribute xmi:type="uml:Property" name="VALOR”>
Modelio	ownedLiteral	uml:EnumerationLiteral	name	<uml:Model> <packagedElement xmi:type="uml:Enumeration" name="NOMBRE"> <ownedLiteral xmi:type="uml:EnumerationLiteral" name="VALOR"/>
Rational Software Architect	ownedLiteral	uml:EnumerationLiteral	name	<uml:Package> <packagedElement xmi:type="uml:Enumeration" name="NOMBRE"> <ownedLiteral xmi:type="uml:EnumerationLiteral" name="VALOR">
Visual Paradigm	ownedLiteral	uml:EnumerationLiteral	name	<xmi:XMI> <uml:Model> <ownedMember name="NombreDeEnumeracion" xmi:type="uml:Enumeration"> <ownedLiteral name="ValorDeEnumeracion" xmi:type="uml:EnumerationLiteral">
MagicDraw Enterprise	ownedAttribute	uml:Property	name	<xmi:XMI> <uml:Model name="NOMBREdelMODELO"> <packagedElement> <nestedClassifier xmi:type="uml:Enumeration" name="NOMBREdeENUMERACION"> <ownedAttribute xmi:type="uml:Property" name="VALORdeENUMERACION">

Tabla 17. Comparativa de XMI para el estereotipo Descriptor en las propiedades de una Clase

Atributo de Clase Descriptor			
Herramienta	**Contenedor**	**Tipo XMI**	**Arbol XMI**
Enterprise Architect	thecustomprofile:descriptor	base_Attribute indica el xmi:id del atributo de clase al cual aplica.	<xmi:XMI> <uml:Model xmi:type="uml:Model"> <packagedElement xmi:type="uml:Package"> <packagedElement xmi:type="uml:Package"> <packagedElement xmi:type="uml:Class"> <ownedAttribute xmi:type="uml:Property" xmi:id="ID" name="NOMBREdeATRIBUTO"> Con: <xmi:XMI> <uml:Model xmi:type="uml:Model"> thecustomprofile:descriptor base_Attribute="ID">
Modelio	LocalProfile:descriptor	base_Property indica el xmi:id del atributo de clase al cual aplica.	<xmi:XMI> <uml:Model> <packagedElement xmi:type="uml:Class" name="NOMBRE"> <ownedAttribute xmi:type="uml:Property" xmi:id="ID-DESCRIPTOR" name="ATRIBUTO"> Con : <xmi:XMI> <LocalProfile:descriptor base_Property="ID-DESCRIPTOR"/>
Rational Software Architect	details	key="descriptor"	<uml:Model> <packagedElement xmi:type="uml:Class" name="NOMBREdeCLASE"> <ownedAttribute xmi:type="uml:Property" name="NOMBREdeATRIBUTO"> <xmi:Extension> <eAnnotations> <details key="descriptor">
Visual Paradigm	appliedStereotype	value="Attribute_descriptor_id"	<xmi:XMI> <uml:Model> <ownedMember name="NOMBRE" xmi:type="uml:Class"> <ownedAttribute name="NombreAtributo" xmi:type="uml:Property"> <xmi:Extension> <attribute/> <appliedStereotype xmi:value="Attribute_descriptor_id">
MagicDraw Enterprise	Data:descriptor	base_Element indica el xmi:id del atributo de clase al cual aplica.	<xmi:XMI> <uml:Model name="NOMBREdelMODELO"> <packagedElement> <nestedClassifier xmi:type="uml:Class" name="NOMBREdeCLASE"> <ownedAttribute xmi:id="ID" name="NOMBREdeATRIBUTO"> Con : <xmi:XMI> <Data:descriptor base_Element="ID">

Tabla 18. Comparativa de XMI para el estereotipo Identifier en las propiedades de una Clase

Atributo de Clase Identifier			
Herramienta	**Contenedor**	**Tipo XMI**	**Arbol XMI**
Enterprise Architect	thecustomprofile:identifier	base_Attribute indica el xmi:id del atributo de clase al cual aplica.	<xmi:XMI> <uml:Model xmi:type="uml:Model"> <packagedElement xmi:type="uml:Package"> <packagedElement xmi:type="uml:Package"> <packagedElement xmi:type="uml:Class"> <ownedAttribute xmi:type="uml:Property" xmi:id="ID" name="NOMBREdeATRIBUTO"> Con: <xmi:XMI> <uml:Model xmi:type="uml:Model"> thecustomprofile:identifier base_Attribute="ID">
Modelio	LocalProfile:identifier	base_Property indica el xmi:id del atributo de clase al cual aplica.	<xmi:XMI> <uml:Model> <packagedElement xmi:type="uml:Class" name="NOMBRE"> <ownedAttribute xmi:type="uml:Property" xmi:id="ID-IDENTIFIER" name="ATRIBUTO"> Con : <xmi:XMI> <LocalProfile:identifier base_Property="ID-IDENTIFIER"/>
Rational Software Architect	details	key="identifier"	<uml:Model> <packagedElement xmi:type="uml:Class" name="NOMBREdeCLASE"> <ownedAttribute xmi:type="uml:Property" name="NOMBREdeATRIBUTO"> <xmi:Extension> <eAnnotations> <details key="identifier">
Visual Paradigm	appliedStereotype	value="Attribute_identifier_id"	<xmi:XMI> <uml:Model> <ownedMember name="NOMBRE" xmi:type="uml:Class"> <ownedAttribute name="NombreAtributo" xmi:type="uml:Property"> <xmi:Extension> <attribute/> <appliedStereotype xmi:value="Attribute_identifier_id">
MagicDraw Enterprise	Data:identifier	base_Element indica el xmi:id del atributo de clase al cual aplica.	<xmi:XMI> <uml:Model name="NOMBREdelMODELO"> <packagedElement> <nestedClassifier xmi:type="uml:Class" name="NOMBREdeCLASE"> <ownedAttribute xmi:id="ID" name="NOMBREdeATRIBUTO"> Con : <xmi:XMI> <Data:identifier base_Element="ID">

Tabla 19.Comparativa de XMI para tipos de datos no estándar

Tipo de dato Generico			
Herramienta	**Contenedor**	**Tipo XMI**	**Arbol XMI**
Enterprise Architect	properties	type=”TipoDeDato”	<xmi:XMI> <xmi:Extension> <elements> <element xmi:type=”uml:Class” name”NombreDeClase”> <attributes> <attribute name =”NombreDeAtributo”> <properties type=”TipoDeDato”>
Modelio	packagedElement	xmi:id indica el xmi:id del atributo de clase al cual aplica.	<xmi:XMI> <uml:Model> <packagedElement xmi:type="uml:Class" name="NOMBRE"> <ownedAttribute xmi:type="uml:Property" name="ATRIBUTO" type=" ID-ADDRESS”> Con : <xmi:XMI> <uml:Model> <packagedElement xmi:type="uml:PrimitiveType" xmi:id="ID-ADDRESS" name="TIPOdeDATO">
Rational Software Architect	packagedElement	xmi:id indica el xmi:id del atributo de clase al cual aplica.	<uml:Model> <packagedElement xmi:type="uml:Class" name="NOMBREdeCLASE"> <ownedAttribute xmi:type="uml:Property" name="NOMBREdeATRIBUTO" type="ID"> Con: <uml:Model> <packagedElement xmi:type="uml:DataType" xmi:id=”ID" name="TIPOdeDATO">
Visual Paradigm	ownedAttribute	uml:Property	<xmi:XMI> <uml:Model name=”NOMBREdelMODELO”> <ownedMember name=”NombreDeClase” xmi:type=”uml:Class”> <ownedAttribute name="NombreDeAttributo" type="Tipo_id" xmi:type="uml:Property">
MagicDraw Enterprise	packagedElement	xmi:id indica el xmi:id del atributo de clase al cual aplica.	<xmi:XMI> <uml:Model name="NOMBREdelMODELO"> <packagedElement> <nestedClassifier xmi:type="uml:Class" name="NOMBREdeCLASE"> <ownedAttribute xmi:type="uml:Property" type="IdTIPOdeDATO"> Con: <xmi:XMI> <uml:Model name="NOMBREdelMODELO"> <packagedElement xmi:type="uml:Class" xmi:id=”IdTIPOdeDATO" name="NOMBREdeTIPOdeDATO">

Tabla 20. Comparativa de XMI para Componentes UML

Componente				
Herramienta	**Contenedor**	**Tipo XMI**	**Nombre**	**Arbol XMI**
Enterprise Architect	packagedElement	uml:Component	name	<xmi:XMI> <uml:Model> <packagedElement xmi:type="uml:Package"> <packagedElement xmi:type="uml:Component" name="NOMBREdeCOMPONENTE">
Modelio	packagedElement	uml:Component	name	<uml:Model> <packagedElement xmi:type="uml:Component" name="NOMBREdelCOMPONENTE"/>
Rational Software Architect	packagedElement	uml:Component	name	<uml:Package> <packagedElement xmi:type="uml:Component" name="NOMBREdeCOMPONENTE">
Visual Paradigm	ownedMember	uml:Component	name	<xmi:XMI> <uml:Model name="NOMBREdelMODELO"> <ownedMember name="NOMBREdelCOMPONENTE" xmi:type="uml:Component">
MagicDraw Enterprise	packagedElement	uml:Component	name	<xmi:XMI> <uml:Model name="NOMBREdelMODELO"> <packagedElement xmi:type="uml:Component" name="NOMBREdeCOMPONENTE">

Tabla 21. Comparativa de XMI para estereotipos que identifica los tipos de Componentes

Tipo de Componente			
Herramienta	**Contenedor**	**Tipo XMI**	**Arbol XMI**
Enterprise Architect	properties	stereotype	<xmi:XMI> <xmi:Extension> <elements> <element xmi:type”uml:Component” name=”NombreDeComponente”> <properties stereotype=”TipoDeComponente”>
Modelio	LocalProfile:TIPOdeCOMPONENTE	base_Component indica el xmi:id del atributo de clase al cual aplica.	<xmi:XMI <uml:Model> <packagedElement xmi:type="uml:Component" xmi:id="ID-COMPONENTE" name="NOMBRE"/> Con : <xmi:XMI> <LocalProfile:TIPOdeCOMPONENTE base_Component="ID-COMPONENTE"/>
Rational Software Architect	details	key	<uml:Package> <packagedElement xmi:type="uml:Component" name="NOMBREdeCOMPONENTE"> <xmi:Extension> <eAnnotations> <details key="TIPOdeCOMPONENTE">
Visual Paradigm	appliedStereotype	xmi:value="TipoDeComponente_id"	<xmi:XMI> <uml:Model> <ownedMember name="NombreDeComponente" xmi:type="uml:Component"> <xmi:Extension> <appliedStereotype xmi:value="TipoDeComponente_id">
MagicDraw Enterprise	Data:TIPOdeCOMPONENTE	base_Element indica el xmi:id del atributo de clase al cual aplica.	<xmi:XMI> <uml:Model name="NOMBREdelMODELO"> <packagedElement xmi:type="uml:Component" xmi:id="ID" name="NOMBREdeCOMPONENTE"/> Con: <xmi:XMI> <Data:TIPOdeCOMPONENTE base_Element="ID">

Tabla 22. Comparativa de XMI para valores etiquetados

Valores Etiquetados				
Herramienta	**Contenedor**	**Tipo XMI**		**Arbol XMI**
Enterprise Architect	thecustomprofile:ETIQUETA	base_Component indica el xmi:id del atributo de clase al cual aplica.		<xmi:XMI> <uml:Model> <thecustomprofile:ETIQUETA ETIQUETA="VALOR"/>
Modelio	default:Classifier	base_Classifier indica el xmi:id del atributo de clase al cual aplica.		<xmi:XMI <uml:Model> <packagedElement xmi:type=" uml:Component " xmi:id="ID" name="NOMBRE"> Con : <xmi:XMI <default:Classifier base_Classifier="ID"> <persistence>VALOR</persistence>
Rational Software Architect	ownedAttribute	uml:Property	name	<uml:Package> <packagedElement name="NOMBREdelELEMENTO"> <ownedAttribute xmi:type="uml:Property" name="NOMBREdeTV"> <defaultValue> <language/> <body>VALORdelTV</body>
Visual Paradigm	property	string	name	<xmi:XMI> <uml:Model> <xmi:Extension> <vpumlChildModels> <vpumlModel modelType="DefaultTaggedValueContainer"> <vpumlChildModels> <vpumlModel modelType="DefaultTaggedValue"> <properties> <property name="name" type="string" value="ETIQUETA">
MagicDraw Enterprise	ownedAttribute	uml:Property	name	<xmi:XMI> <uml:Model name="NOMBREdelMODELO"> <packagedElement> <nestedClassifier> <ownedAttribute xmi:type="uml:Property" name="NOMBREdelTV"> <defaultValue xmi:type="uml:LiteralString" value="VALORedelTV"/>

ANEXO B – Modelado de aplicaciones

En este apartado se muestran algunos modelados adicionales a los ya nombrados en el cuerpo de la tesis. Esto permite comprobar la flexibilidad de la metodología y su aplicación a distintos dominios y tipos de aplicaciones.

B1. Compra Venta de Productos

La aplicación consiste en un sistema para compra y venta de productos. Un usuario puede publicar productos para vender en distintas categorías; también puede realizar búsquedas y comprar productos publicados por otros usuarios.

La Figura 40 muestra el modelo de datos de la aplicación, donde se definen los usuarios, las categorías, los productos y los estados que puede tomar un producto.

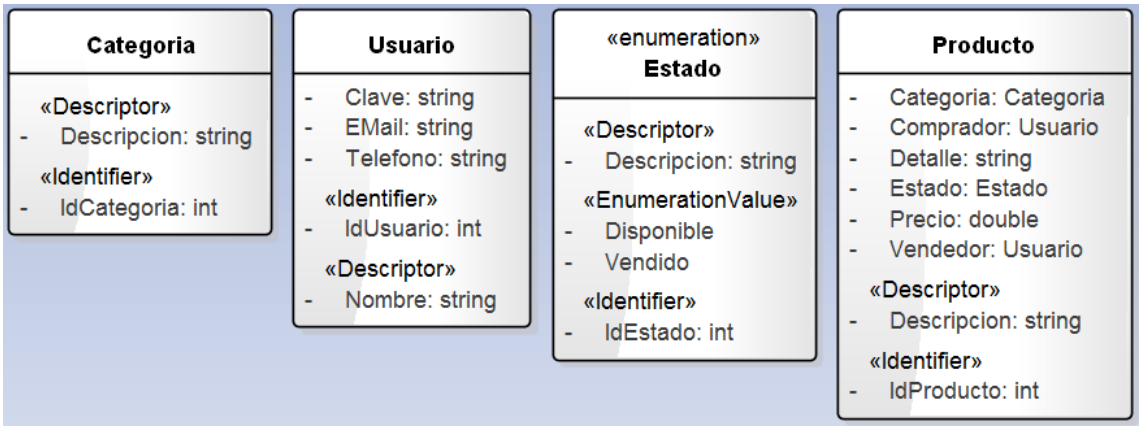

Figura 40. Modelo de datos para la aplicación de compra-venta

El modelo de Navegación e Interfaz de usuario está compuesto por 9 componentes que se pueden ver en la Figura 41.

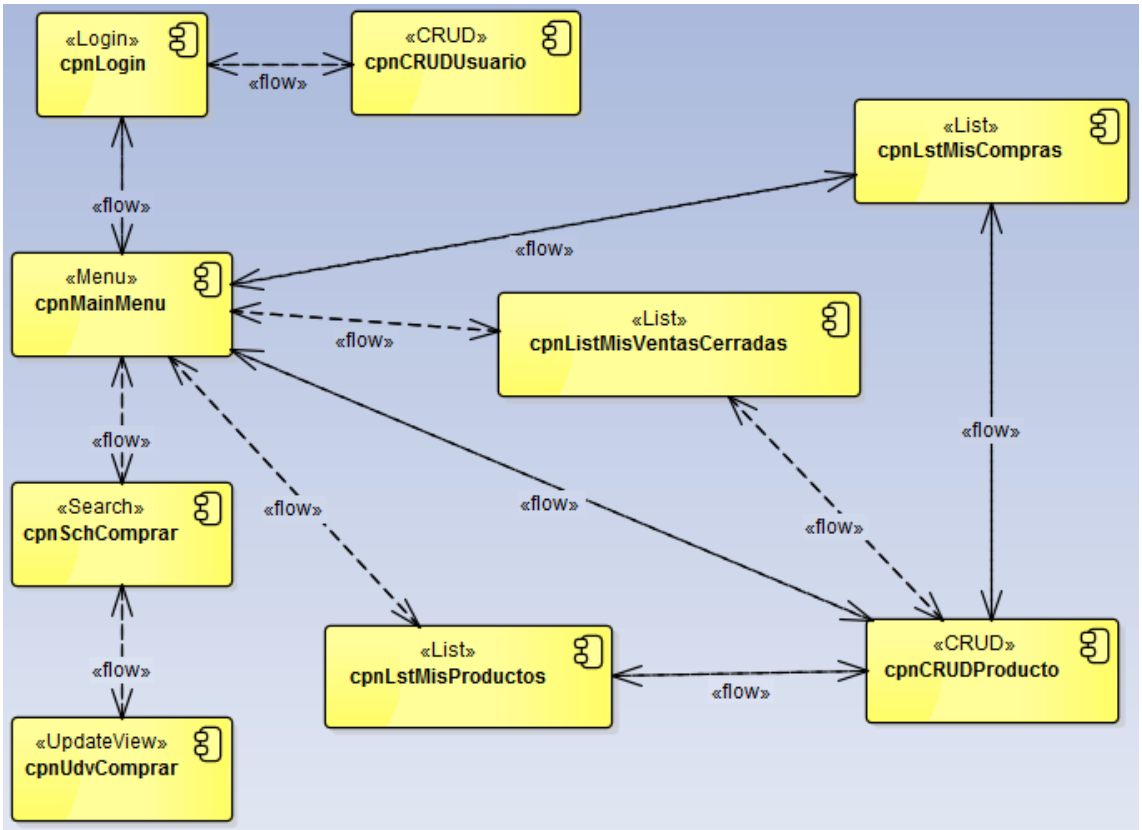

Figura 41. Modelo de Interfaz de Usuario del sistema de Compra-Venta

Desde el componente Login se puede acceder al sistema o crear un nuevo usuario si es que no se dispone de uno. La configuración de este componente está detallada en la Tabla 23.

Tabla 23. Valores etiquetados del Login de Compra-Venta

Etiqueta	Valor
Id	cpnLogin
Title	Compra Venta
MainEntity	Usuario
RedirectToComponent	cpnMainMenu
User	Nombre
Password	Clave
SelectableUser	False
Navigation	Link("Crear Usuario",cpnCRUDUsuario,C,,)

La Figura 42 muestra la pantalla generada a partir de este componente.

Figura 42. Pantalla de login del sistema de Compra-Venta

Si se selecciona la opción "Crear Usuario", se muestra un componente del tipo CRUD configurado para crear un nuevo usuario. La configuración de dicho componente puede verse en la Tabla 24 y la pantalla generada a partir del mismo en la Figura 43.

Tabla 24. Valores etiquetados para el componente de creación de nuevos usuarios

Etiqueta	Valor
Id	cpnCRUDUsuario
MainEntity	Usuario
Title	Edición de Usuario
Navigation	Link("Volver",cpnLogin,,,)
DefaultValuesCreate	
DefaultValuesUpdate	
SkippedPropertiesCreate	
SkippedPropertiesUpdate	
CreateEntityOnUpdate	
CreatEntityOnCreate	
OptionalPropertiesCreate	
OptionalPropertiesUpdate	
RequiresAuthentication	False
FilterRelatedPropertiesByLoggedUser	False

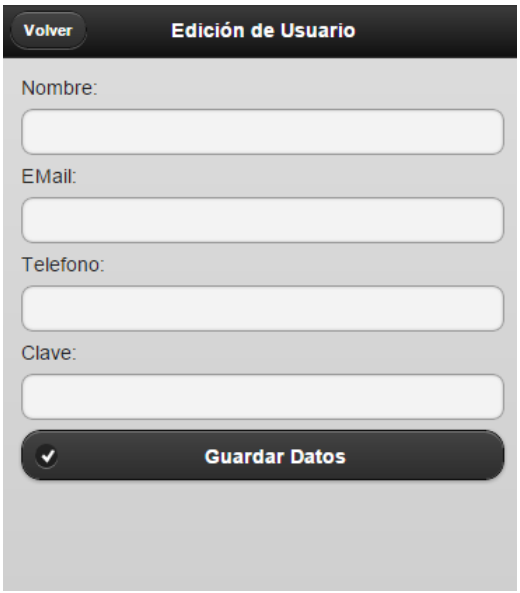

Figura 43. Pantalla de creación de nuevos usuarios

Una vez logueado exitosamente el usuario es redireccionado al menú principal de la aplicación donde verá las distintas acciones que puede realizar en el sistema. Dicho componente se configura mediante los valores etiquetados de la Tabla 25.

Tabla 25. Valores etiquetados del componente tipo `Menu` del sistema de Compra-Venta

Etiqueta	Valor
Id	cpnMainMenu
Title	Menu Principal
Options	Link("Comprar",cpnSchComprar,,,), Link("Vender",cpnCRUDProducto,C,,), Link("Mis Productos",cpnLstMisProductos,,,), Link("Mis Ventas Cerradas",cpnListMisVentasCerradas,,,), Link("Mis Compras",cpnLstMisCompras,,,)
Navigation	Link("LogOut",cpnLogin,,,)
RequiresAuthentication	True

La pantalla generada a partir del componente menú puede verse en la Figura 44.

Figura 44. Menú principal del sistema de compra-venta.

La primera opción del menú permite comprar un producto, para lo cual será necesario visualizar los productos en venta, cargados por otros usuarios. Es decir, se visualizaran productos en estado "Disponible" y además que hayan sido creados por un usuario distinto al logueado actualmente. Es por eso que se utiliza un componente del tipo Search, el cual mediante filtros permite encontrar el producto deseado. La configuración completa de dicho componente puede verse en la Tabla 26 y la pantalla generada en la Figura 45.

Tabla 26. Configuración del componente para buscar productos a comprar.

Etiqueta	Valor
Id	cpnSchComprar
MainEntity	Producto
Title	Buscar Producto
Navigation	Link("Home",cpnMainMenu,,,)
FixedFilters	Producto.Estado = Estado.Disponible AND Producto.Vendedor <> LOGGEDUSER
Columns	Descripcion;
AditionalInformationLine	Categoria,Precio
DefaultAction	Link("Comprar",cpnUdvComprar,,,)
Actions	
Sort	Descripcion ASC
SearchFilters	Descripcion FreeText,Categoria SingleSelection
RequiresAuthentication	True
FilterRelatedPropertiesByLoggedUser	True

Figura 45. Pantalla de búsqueda de productos

Al seleccionar un producto del listado resultado de la búsqueda, el usuario será llevado a una pantalla donde puede ver más información y confirmar su compra. Esta pantalla se configura mediante un componente del tipo `UpdateView` que actualizará el producto cambiando su estado y guardando como comprador el usuario logueado. La configuración de los valores etiquetados del componente puede verse en la Tabla 27 y la pantalla generada en la Figura 46.

Tabla 27. Valores etiquetados del componente para confirmar una compra

Etiqueta	Valor
Id	cpnUdvComprar
MainEntity	Producto
Title	Comprar
Navigation	Link("Volver",cpnSchComprar,,,), Link("Home",cpnMainMenu,,,)
DisplayProperties	Descripcion,Detalle,Precio
UpdateProperties	
CreateEntity	
DefaultValuesUpdate	Producto.Comprador=LOGGEDUSER, Producto.Estado=Estado.Vendido
RequiresAuthentication	True
FilterRelatedPropertiesByLoggedUser	True
DisplayProperties	Descripcion,Detalle,Precio

Figura 46. Pantalla de confirmación de compra

La opción vender llevará a una pantalla para crear un nuevo producto, donde el vendedor será automáticamente completado con el usuario logueado y el estado será "Disponible". El componente tipo `CRUD` para vender un producto se configura con los valores etiquetados de la Tabla 28 y la Figura 47 muestra la pantalla generada a partir del mismo.

Tabla 28. Configuración del componente para vender un nuevo producto

Etiqueta	Valor
Id	cpnCRUDProducto
MainEntity	Producto
Title	Edición de Producto
Navigation	Link("Volver",BACK,,,), Link("Home",cpnMainMenu,,0,);)
DefaultValuesCreate	Producto.Estado=Estado.Disponible, Producto.Vendedor=LOGGEDUSER
DefaultValuesUpdate	
SkippedPropertiesCreate	Comprador
SkippedPropertiesUpdate	Vendedor,Estado,Comprador
CreateEntityOnUpdate	
CreatEntityOnCreate	
OptionalPropertiesCreate	
OptionalPropertiesUpdate	
RequiresAuthentication	True
FilterRelatedPropertiesByLoggedUser	True

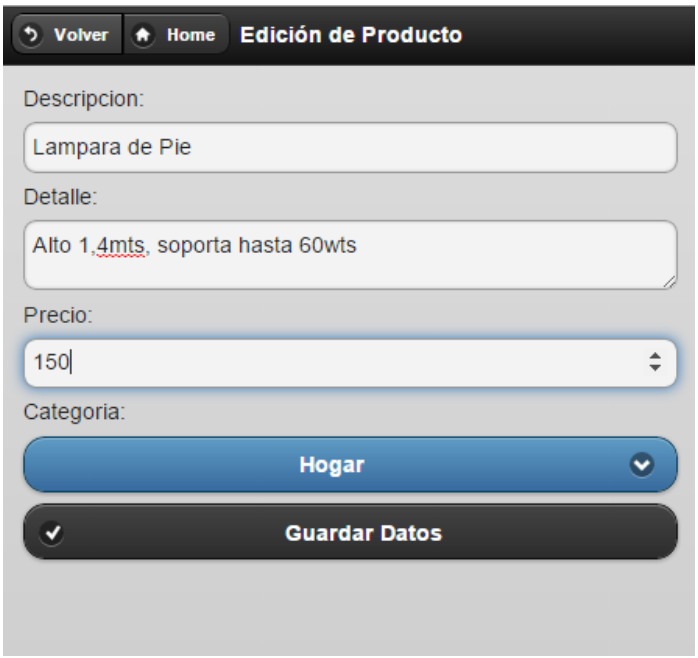

Figura 47. Pantalla para vender un nuevo producto

Las últimas tres opciones del menú principal, corresponden a listados de productos pre-configurados para mostrar rápidamente información la que será visualizada en forma de grilla tal como la mostrada en el resultado de la búsqueda al comprar un producto. El primer listado corresponde a los productos del usuario logueado que aún están en venta y a partir del cual se puede acceder a modificar un producto o eliminarlo (la configuración puede verse en la Tabla 29). El segundo listado muestra productos del usuario logueado que ya fueron vendidos y su correspondiente configuración se muestra en la Tabla 30. Las ventas cerradas no pueden modificarse, por lo que solo se podrá acceder al detalle de los productos en modo consulta. El último listado se corresponde a las compras realizadas por el usuario logueado cuya configuración se muestra en la Tabla 31.

Tabla 29. Configuración del listado de productos a la venta

Etiqueta	Valor
Id	cpnLstMisProductos
MainEntity	Producto
Title	Listado de Producto
Navigation	Link("Home",cpnMainMenu,,0,);)
Columns	Descripcion;
DefaultAction	Link("Editar",cpnCRUDProducto,UD,1,),
Sort	Descripcion ASC
FixedFilters	Producto.Vendedor = LOGGEDUSER AND Producto.Estado = Disponible
AditionalInformationLine	Categoria,Precio
Actions	
RequiresAuthentication	True

Tabla 30. Listado para visualizar productos vendidos.

Etiqueta	Valor
Id	cpnListMisVentasCerradas
MainEntity	Producto
Title	Ventas Cerradas
Navigation	Link("Home",cpnMainMenu,,,)
FixedFilters	Producto.Estado = Estado.Vendido AND Producto.Vendedor = LOGGEDUSER
Columns	Descripcion;
AditionalInformationLine	Categoria,Precio
DefaultAction	Link("Consultar",cpnCRUDProducto,R,,)
Actions	
Sort	Descripcion ASC
RequiresAuthentication	True

Tabla 31. Listado para visualizar compras realizadas.

Etiqueta	Valor
Id	cpnLstMisCompras
MainEntity	Producto
Title	Mis Compras
Navigation	Link("Home",cpnMainMenu,,,)
FixedFilters	Producto.Comprador = LOGGEDUSER
Columns	Descripcion;
AditionalInformationLine	Categoria,Precio
DefaultAction	Link("Visualizar",cpnCRUDProducto,R,,)
Actions	
Sort	Descripcion ASC
RequiresAuthentication	True

B2. Log de Viajes

Se desea modelar un sistema que permita a un viajero registrar los eventos importantes de un viaje, marcando diferentes lugares y haciendo comentarios sobre los mismos. Cada usuario podrá administrar sus lugares y además al momento de registrar un evento (denominado hito) se debe poder utilizar las capacidades de geo-localización del dispositivo móvil para tomar en forma automática las coordenadas exactas de la ubicación del hito.

El modelo de datos consta de tres clases que pueden verse en la Figura 48. Una clase Viajero que representa los distintos usuarios del sistema; la clase Lugar donde cada viajero podrá administrar los distintos lugares que visite y por último la clase Hito que será donde se registren las referencias que el viajero desee guardar sobre el lugar que está visitando.

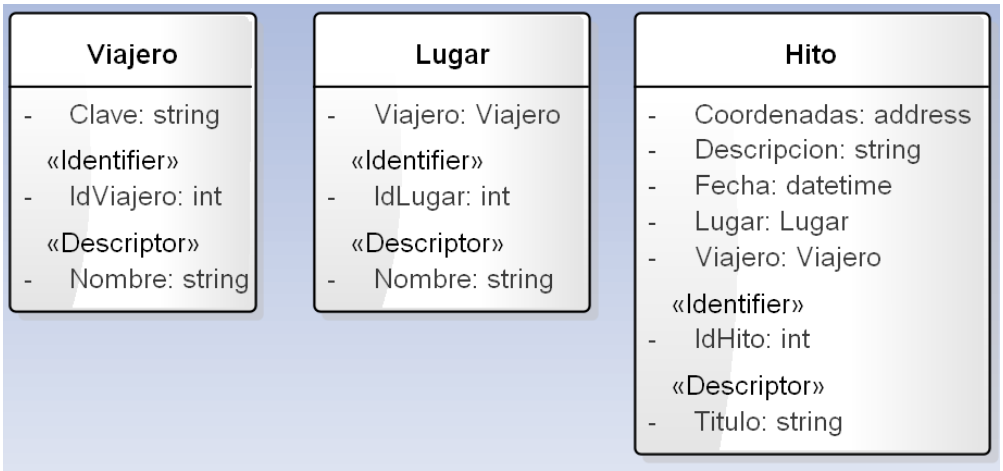

Figura 48. Modelo de datos para el Log de Viajes

Para diseñar la interfaz de usuario se utilizaron 8 componentes, tal como muestra la Figura 49.

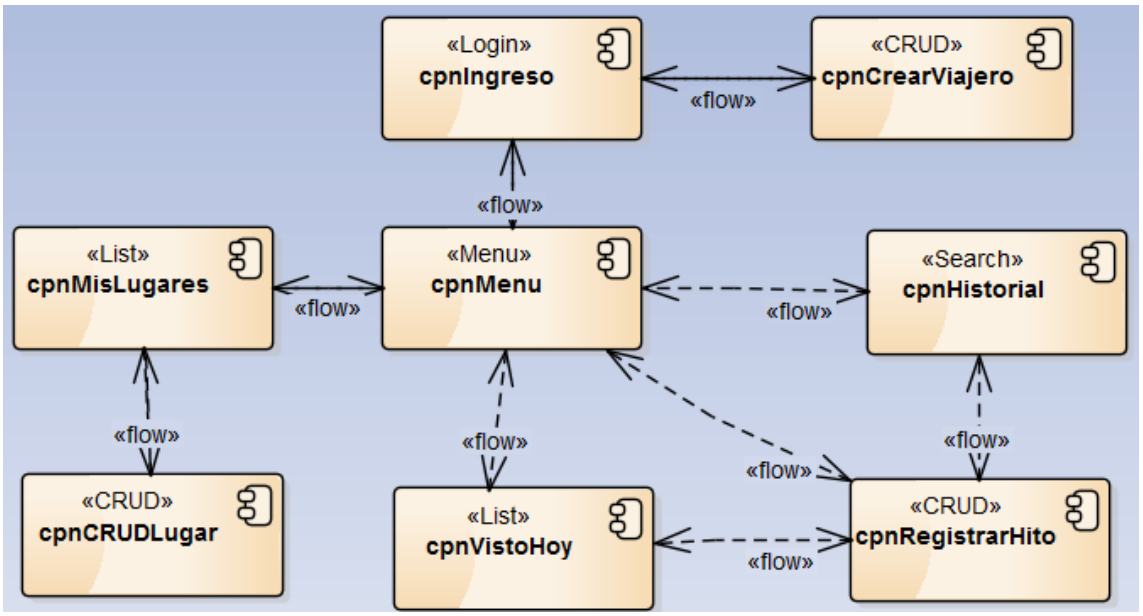

Figura 49. Componentes para el sistema de log de viajes

El sistema guarda información por viajero por lo tanto para utilizarlo es necesario primero autenticarse mediante un componente del tipo `Login`. En el mismo se agrega un link para que los nuevos viajeros puedan registrarse. La configuración de este componente se realiza mediante los valores etiquetados de la Tabla 32. El componente para registrarse nuevos viajeros se realiza mediante un componente del tipo `CRUD` configurado según la Tabla 33. Este componente es llamado desde el login con el parámetro "`C`" para que solo permita crear nuevos registros. La Figura 50 muestra a la izquierda la pantalla de login y a la derecha la pantalla para crear nuevos usuarios generados de forma automática con estos dos componentes.

Tabla 32. Configuración del componente Login para el Log de Viajes

Etiqueta	Valor
Id	cpnIngreso
MainEntity	Viajero
Title	Log de Viajes
RedirectToComponent	cpnMenu
User	Nombre
Password	Clave
SelectableUser	False
Navigation	Link("Registrarse",cpnCrearViajero,C,,)

Tabla 33. Valores etiquetados del componente para crear un viajero

Etiqueta	Valor
Id	cpnCrearViajero
MainEntity	Viajero
Title	Crear Viajero
Navigation	Link("Volver",BACK,,,)
DefaultValuesCreate	
DefaultValuesUpdate	
SkippedPropertiesCreate	
SkippedPropertiesUpdate	
CreateEntityOnUpdate	
CreatEntityOnCreate	
OptionalPropertiesCreate	
OptionalPropertiesUpdate	
RequiresAuthentication	False
FilterRelatedPropertiesByLoggedUser	False

Figura 50. Pantalla de ingreso al sistema y de creación de un viajero

Luego de loguearse exitosamente el usuario es redireccionado al menú de la aplicación donde dispondrá de distintas opciones. Allí podrá administrar sus lugares, registrar un nuevo hito, consultar y modificar los hitos del día, así como consultar hitos anteriores. La configuración del componente puede verse en la Tabla 34 y la pantalla generada a partir del mismo en la Figura 51 .

Tabla 34. Configuración del Menu Principal del sistema de Log de Viajes

Etiqueta	Valor
Id	cpnMenu
Title	Log de Viajes
Options	Link("Visto Hoy",,cpnVistoHoy,,1,), Link("Mis Lugares",,cpnMisLugares,,2,), Link("Nuevo Hito",,cpnRegistrarHito,C,3,), Link("Historial",,cpnHistorial,,,)
Navigation	Link("Salir",,cpnIngreso,,,)
RequiresAuthentication	True

Figura 51. Menú del sistema de Log de Viajes

Antes de poder cargar un hito el viajero deberá crear al menos un lugar. Para ello dispone la opción Mis lugares, que despliega los lugares creados por el viajero logueado en el sistema. Este componente es del tipo List configurado según la Tabla 35. Desde ese listado el viajero será capaz de crear, modificar y eliminar sus lugares, mediante un componente del tipo CRUD cuya configuración puede verse en la

Tabla 36. A partir de esos componentes se generan en forma automática las pantallas de la Figura 52 .

Tabla 35. Configuración del componente del listado de lugares del viajero logueado

Etiqueta	Valor
Id	cpnMisLugares
MainEntity	Lugar
Title	Lugares
Navigation	Link("Agregar",,cpnCRUDLugar,C,,),

	Link("Home",cpnMenu,,,)
Columns	Nombre;
DefaultAction	Link("Editar",cpnCRUDLugar,UD,,))
Sort	Nombre ASC
FixedFilters	
AditionalInformationLine	
Actions	
RequiresAuthentication	True

Tabla 36. Configuración del componente de administración de Lugares

Etiqueta	Valor
Id	cpnCRUDLugar
MainEntity	Lugar
Title	Editar Lugar
Navigation	Link("Home", cpnMenu,,,)
DefaultValuesCreate	Lugar.Viajero=LOGGEDUSER
DefaultValuesUpdate	
SkippedPropertiesCreate	
SkippedPropertiesUpdate	Viajero
CreateEntityOnUpdate	
CreatEntityOnCreate	
OptionalPropertiesCreate	
OptionalPropertiesUpdate	
RequiresAuthentication	True
FilterRelatedPropertiesByLoggedUser	True

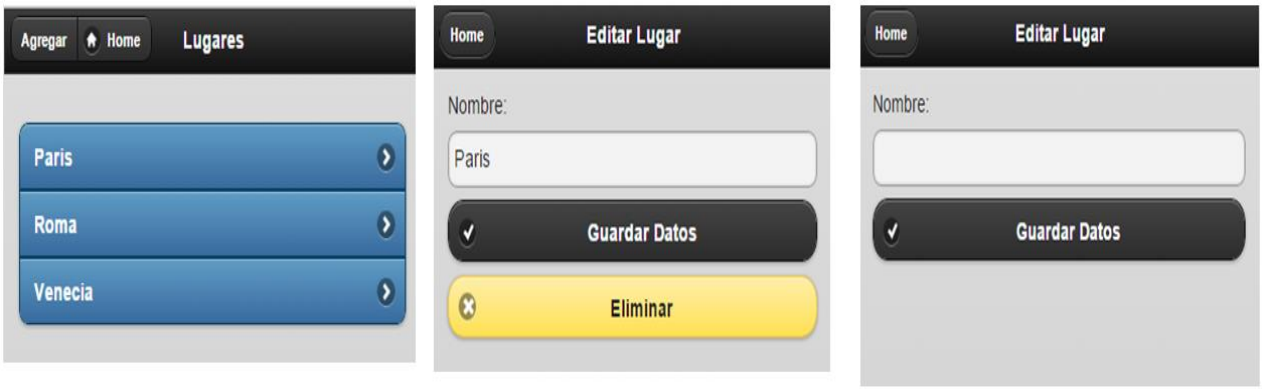

Figura 52. Pantallas para la administración de Lugares del viajero logueado

Ahora sí, el viajero puede comenzar a registrar los distintos hitos, para ello dispone de una opción directamente desde el menú principal que es “Nuevo Hito”. Esta pantalla es generada mediante un componte del tipo `CRUD` configurado con los valores etiquetados de la Tabla 37 . En la Figura 53 puede verse la pantalla de creación de un Hito donde debido a que el campo coordenadas fue configurado con el tipo de datos `address` se utilizaron las características de

geo-localización del dispositivo para recuperar de forma automática la posición exacta del viajero en ese momento.

Tabla 37. Configuración del componente para registrar Hitos

Etiqueta	Valor
Id	cpnRegistrarHito
MainEntity	Hito
Title	Hito
Navigation	Link("Volver",BACK,,,), Link("Home",cpnMenu,,,)
DefaultValuesCreate	Hito.Viajero=LOGGEDUSER, Hito.Fecha=NOW
DefaultValuesUpdate	
SkippedPropertiesCreate	
SkippedPropertiesUpdate	Viajero, Fecha, Coordenadas
CreateEntityOnUpdate	
CreatEntityOnCreate	
OptionalPropertiesCreate	Coordenadas
OptionalPropertiesUpdate	
RequiresAuthentication	True
FilterRelatedPropertiesByLoggedUser	True

Figura 53. Pantalla de creación de un nuevo hito

Una vez registrados los hitos, el viajero dispone de dos listados para consultar la información. El primer listado, configurado según la Tabla 38, muestra todos los hitos registrados en ese día, permitiendo modificarlos o eliminarlos. El segundo listado es un componente del tipo `Search` que permite consultar los hitos históricos aplicando distintos filtros sobre los campos, como fecha, lugar o título. Al pulsar sobre un ítem de la grilla del resultado de la búsqueda se accede a visualizar la información del hito no pudiendo modificarla. Este componente se configura con los valores etiquetados de la Tabla 39. En la Figura 54 puede verse a la izquierda el listado de Hitos del día y a la derecha la pantalla del historial con los distintos filtros de búsqueda y el listado resultante.

Tabla 38. Valores etiquetados de listado de hitos del día

Etiqueta	Valor
Id	cpnVistoHoy
MainEntity	Hito
Title	Hitos de Hoy
Navigation	Link("Home",cpnMenu,,,)
Columns	Titulo;
DefaultAction	Link("Editar",cpnRegistrarHito,UD,,)
Sort	Fecha DESC, Titulo ASC
FixedFilters	Hito.Fecha = TODAY AND Hito.Viajero = LOGGEDUSER
AditionalInformationLine	Fecha, Lugar
Actions	
RequiresAuthentication	True

Tabla 39. Configuración del componte de búsqueda histórica de hitos

Etiqueta	Valor
Id	Historial
MainEntity	Hito
Title	Historial
Navigation	Link("Home",cpnMenu,,,)
FixedFilters	Hito.Viajero = LOGGEDUSER
Columns	Hito.Titulo;
AditionalInformationLine	Fecha, Lugar
DefaultAction	Link("Consultar",cpnRegistrarHito,R,,)
Actions	
Sort	Fecha DESC
SearchFilters	Titulo FreeText, Lugar SingleSelection, Fecha DateRange
RequiresAuthentication	True
FilterRelatedPropertiesByLoggedUser	True

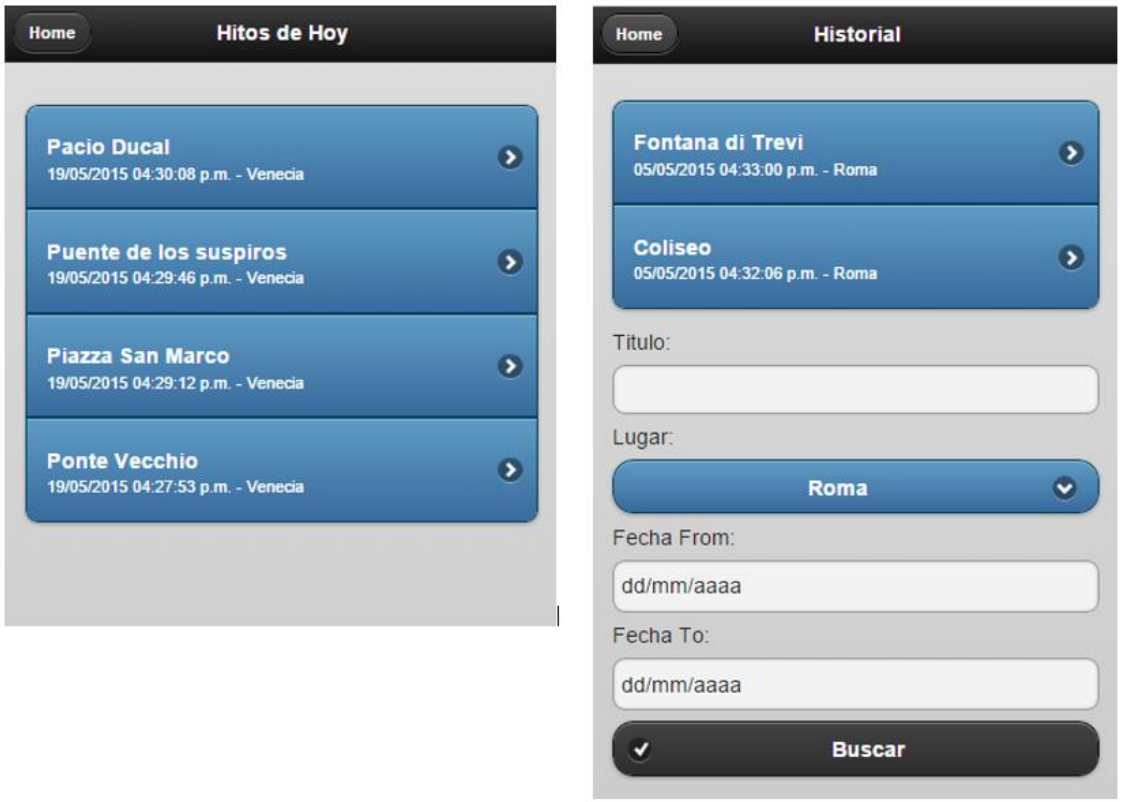

Figura 54. Listados para consulta de hitos registrados

B3. Historia Clínica

Se desea modelar un sistema mediante el cual los médicos puedan administrar las historias clínicas de sus pacientes. Cada médico administra sus propios pacientes y va registrando cada una de las consultas que realiza, informando la fecha, motivo y un diagnóstico detallado. Las sucesivas consultas hacen la historia clínica del paciente donde el médico rápidamente puede revisar el historial para detectar posibles patologías.

Para llevar a cabo este sistema primero se diseña el modelo datos, conformado por las tres clases de la Figura 55.

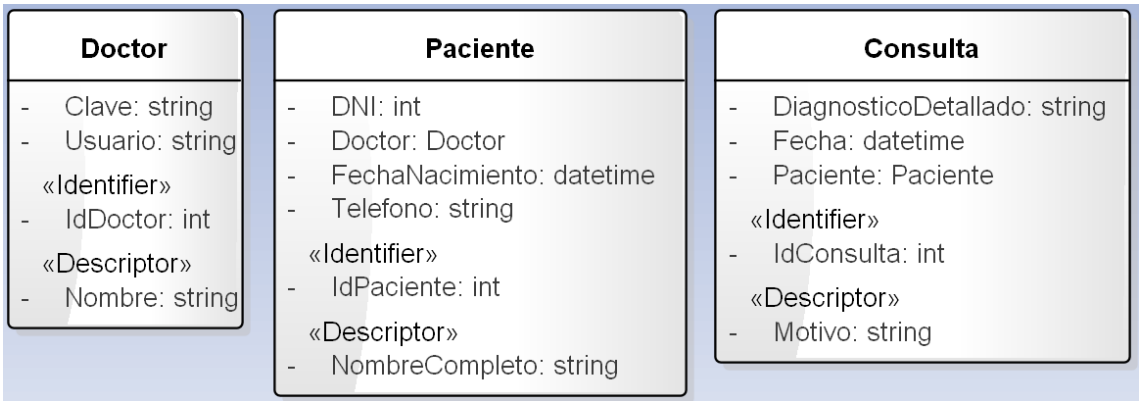

Figura 55. Clases para el sistema de Historia Clínica

La clase Doctor representa los usuarios del sistema que son autenticados mediante un componente del tipo `Login` configurado según los valores etiquetados de la Tabla 40. Este componente dispone además de un link para registrarse en el sistema, de forma que aquellos doctores que no dispongan de una cuenta puedan crear la suya y comenzar a utilizar el sistema. El componente para crear un nuevo Doctor se configura con los valores etiquetados de la Tabla 41. Estos dos componentes generan las pantallas mostradas en la Figura 56.

Tabla 40. Configuración del componente Login para Historia Clínica

Etiqueta	Valor
Id	Ingreso
MainEntity	Doctor
Title	Historia Clinica
RedirectToComponent	BuscarPacientes
User	Usuario
Password	Clave
SelectableUser	False
Navigation	Link("Registrar",cpnCRUDDoctor,C,,)

Tabla 41. Valores etiquetados del componente para crear un nuevo Doctor

Etiqueta	Valor
Id	cpnCRUDDoctor
MainEntity	Doctor
Title	Doctor
Navigation	Link("Volver",BACK,,,)
DefaultValuesCreate	
DefaultValuesUpdate	
SkippedPropertiesCreate	
SkippedPropertiesUpdate	
CreateEntityOnUpdate	
CreatEntityOnCreate	
OptionalPropertiesCreate	
OptionalPropertiesUpdate	
RequiresAuthentication	False
FilterRelatedPropertiesByLoggedUser	False

Figura 56. Pantallas de ingreso y creación de usuarios para el sistema Historia Clínica

Para hacer más sencillo el sistema y que el Doctor pueda acceder directamente a sus pacientes, no se ha definido un menú principal, sino que luego de loguearse exitosamente se mostrará la pantalla de búsqueda de pacientes. Esta pantalla va a permitir además, crear un nuevo paciente, editar y acceder a la historia clínica de pacientes existentes. Para ello se configura un link en la barra de la navegación para el alta de pacientes y en la grilla de resultado de la búsqueda se manejan dos acciones. La acción por defecto llevará a visualizar la historia clínica del paciente, ya que esta será la acción más utilizada. Pero se configura una acción adicional que permitirá modificar el paciente si es que se necesita actualizar algunos de sus datos, como por ejemplo por un cambio de teléfono. En la pantalla se genera entonces por cada ítem resultante de la vista un botón adicional del lado derecho que lleva a la edición del paciente y si se pulsa sobre el lado izquierdo se accede a visualizar la historia clínica. La configuración del componente tipo `search` se muestra en la Tabla 42 y la pantalla generada en la Figura 57.

Tabla 42. Configuración del componente de búsqueda de Pacientes

Etiqueta	Valor
Id	BuscarPacientes
MainEntity	Paciente
Title	Pacientes
Navigation	Link("Salir",Ingreso,,,), Link("Nuevo",cpnCRUDPaciente,C,,)
FixedFilters	Paciente.Doctor = LOGGEDUSER
Columns	Paciente.NombreCompleto;
AditionalInformationLine	DNI
DefaultAction	Link("HistoriaClinica",HistoriaClinica,,,)
Actions	Link("Editar",cpnCRUDPaciente,UD,,)
Sort	NombreCompleto ASC
SearchFilters	NombreCompleto FreeText
RequiresAuthentication	True
FilterRelatedPropertiesByLoggedUser	True

Figura 57. Pantalla de Búsqueda de Pacientes con múltiples acciones sobre el listado resultante

Los Pacientes creados se vinculan automáticamente con el Doctor logueado en el sistema por lo tanto el componte CRUD para editar y modificar paciente tomará por defecto este valor tal como se muestra en la Tabla 43.

Tabla 43. Configuración del componente CRUD para pacientes

Etiqueta	Valor
Id	cpnCRUDDoctor
MainEntity	cpnCRUDPaciente
Title	Paciente
Navigation	Link("Volver",BACK,,,), Link("Home",BuscarPacientes,,0,)
DefaultValuesCreate	Paciente.Doctor=LOGGEDUSER
DefaultValuesUpdate	
SkippedPropertiesCreate	
SkippedPropertiesUpdate	Doctor
CreateEntityOnUpdate	
CreatEntityOnCreate	
OptionalPropertiesCreate	
OptionalPropertiesUpdate	
RequiresAuthentication	True
FilterRelatedPropertiesByLoggedUser	True

Este componente genera una pantalla que permite crear un nuevo Paciente, Modificarlo ó Eliminarlo según los parámetros recibidos. Ejemplos de dichas pantallas pueden verse en la Figura 58.

Figura 58. Pantallas generadas a partir del componente tipo CRUD de pacientes

La historia clínica es un componente del tipo `List` que muestra las consultas realizadas por el paciente seleccionado en la grilla resultante del componente de búsqueda. Es por eso que el listado incorpora un filtro por defecto que se refiere al ítem seleccionado en la pantalla anterior. Esto se logra gracias a la varia especial INVOKEID. La configuración completa del listado puede verse en la Tabla 44 que además permite editar una consulta anterior o crear un registro de una nueva consulta.

Tabla 44. Configuración del componente para visualizar la historia clínica de un paciente

Etiqueta	Valor
Id	HistoriaClinica
MainEntity	Consulta
Title	Historia Clinica
Navigation	Link("Agregar",cpnCRUDConsulta,C,1,), Link("Home",BuscarPacientes,,,)
Columns	Motivo;
DefaultAction	Link("Editar",cpnCRUDConsulta,UD,1,),
Sort	Motivo ASC
FixedFilters	Consulta.Paciente = INVOKEID
AditionalInformationLine	Fecha
Actions	
RequiresAuthentication	True

Este componente genera la pantalla que muestra en Figura 59.

Figura 59. Visualización de la historia clínica de un paciente

El último componente que queda por definir es aquel que permite registrar los datos de la consulta. Esto se realiza mediante un componente del tipo CRUD que es invocado desde el listado de Historia Clínica tanto como para agregar uno nuevo como para visualizar, editar o eliminar uno existente. La configuración de dicho componente puede verse en la Tabla 45 y las pantallas generadas a partir del mismo en la Figura 60.

Tabla 45. Configuración del componente tipo CRUD para consultas

Etiqueta	Valor
Id	cpnCRUDConsulta
MainEntity	Consulta
Title	Consulta
Navigation	Link("Volver",BACK,,,), Link("Home",BuscarPacientes,,,)
DefaultValuesCreate	
DefaultValuesUpdate	
SkippedPropertiesCreate	
SkippedPropertiesUpdate	
CreateEntityOnUpdate	
CreatEntityOnCreate	
OptionalPropertiesCreate	
OptionalPropertiesUpdate	
RequiresAuthentication	True
FilterRelatedPropertiesByLoggedUser	True

Figura 60. Pantallas generadas por el componente tipo CRUD para consultas.

De esta forma se culmina el modelado de la aplicación donde con la utilización de 3 clases y 6 componentes se logran todas las funcionalidades deseadas. La Figura 61 muestra el esquema general de la navegación entre los componentes de la aplicación.

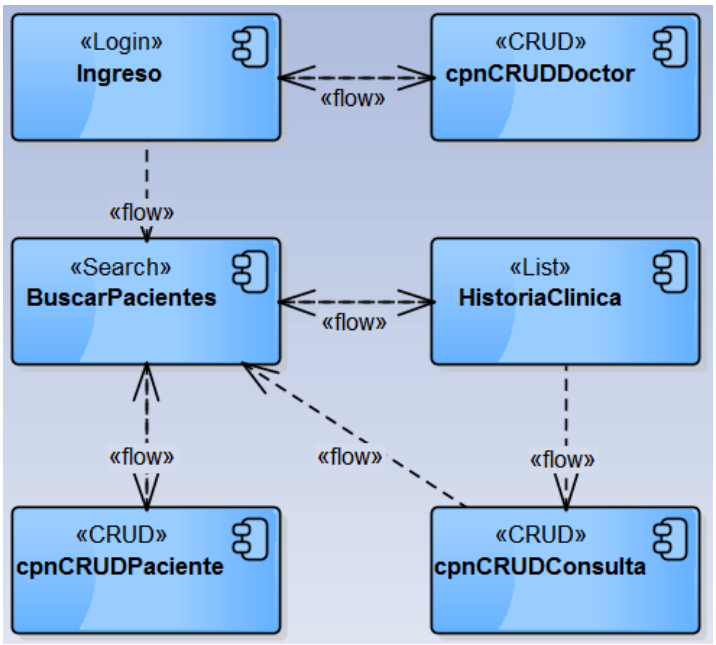

Figura 61. Esquema de Navegación entre los componentes del sistema de Historia Clínica