Universidad Nacional de La Matanza
Departamento de Ingeniería e Investigaciones Tecnológicas

Código: C2-ING-009

Título del Proyecto: Metodología de Modelado para Aplicaciones Móviles - Aplicando MDA y usando Componentes Reutilizables

Programa de Investigación: CyTMA2

Director del Proyecto: Rocío Andrea Rodríguez (RAR)

Co-Director del Proyecto: Daniel Alberto Giulianelli (DAG)

Integrantes del Proyecto:

Docentes: Pablo Martín Vera (PMV), Víctor Manuel Fernández (VMF), Claudia Gabriela Alderete (CGA), Roberto Mariano Ariel Bucher (RMAB)
Alumnos: Anabella Graciela Conca (AGC), Federico Ezequiel Valles (FEV), Javier Gastón Cescon (JGC), Gaspar Acevedo Zain (GAZ)

Fecha de inicio: 2013/01/01 - Fecha de finalización: *2014/12/31*

Resumen: El proyecto plantea la creación de una metodología de modelado y desarrollo para aplicaciones web móviles utilizando el enfoque MDA (Arquitectura dirigida por modelos). Dicha metodología será diseñada mediante una extensión conservativa de UML permitiendo que el modelado pueda realizarse con cualquier herramienta existente en el mercado capaz de exportar sus modelos al formato estándar de intercambio XMI (Intercambio de Metadatos XML). La metodología estará basada en las cuatro actividades principales del modelado hipermedia: modelado conceptual, modelado de la navegación, diseño de la interfaz abstracta e implementación; pero simplifica el modelado de la interfaz y navegación unificándolos en un único diagrama basado en componentes reutilizables. Dichos componentes serán especialmente diseñados para permitir que se pueda generar fácilmente el código fuente de la aplicación a partir de los mismos, incluyendo todos los detalles necesarios para generar una aplicación completa y funcional sin que se deba luego modificar en forma externa el código fuente de la misma. Para ello se construirá una herramienta de transformación que será la encargada en una primer etapa de realizar una transformación entre modelos y luego basada en los modelos finales poder generar tanto el código fuente de la aplicación como su base de datos. Adicionalmente esta herramienta permitirá la utilización de templates para que un mismo modelo pueda generarse el código fuente en distintos lenguajes. Esta característica permitirá por ejemplo generar una aplicación (frontend) para dispositivos móviles y su sistema de administración (backend) en un sitio web de escritorio estándar.

Palabras claves: MDA, APLICACIÓN, MOVIL, UML, XMI

Área de conocimiento: Ingeniería de Comunicaciones, Electrónica y Control
Código de Área de Conocimiento: 1800
Disciplina: Computación
Código de Disciplina: 1802
Campo de Aplicación: Computación
Código de Campo de Aplicación: 1802

Metodología de Modelado para Aplicaciones Móviles Aplicando MDA y usando Componentes Reutilizables

Resumen

El proyecto plantea la creación de una metodología de modelado y desarrollo para aplicaciones web móviles utilizando el enfoque MDA (Arquitectura dirigida por modelos). Dicha metodología será diseñada mediante una extensión conservativa de UML permitiendo que el modelado pueda realizarse con cualquier herramienta existente en el mercado capaz de exportar sus modelos el formato estándar de intercambio XMI (Intercambio de Metadatos XML). La metodología estará basada en las cuatro actividades principales del modelado hipermedia: modelado conceptual, modelado de la navegación, diseño de la interfaz abstracta e implementación; pero simplifica el modelado de la interfaz y navegación unificándolos en un único diagrama basado en componentes reutilizables. Dichos componentes serán especialmente diseñados para permitir que se pueda generar fácilmente el código fuente de la aplicación a partir de los mismos, incluyendo todos los detalles necesarios para generar una aplicación completa y funcional sin que se deba luego modificar en forma externa el código fuente de la misma. Para ello se construirá una herramienta de transformación que será la encargada en una primer etapa de realizar una transformación entre modelos y luego basada en los modelos finales poder generar tanto el código fuente de la aplicación como su base de datos. Adicionalmente esta herramienta permitirá la utilización de templates para que un mismo modelo pueda generarse el código fuente en distintos lenguajes. Esta característica permitirá por ejemplo generar una aplicación (frontend) para dispositivos móviles y su sistema de administración (backend) en un sitio web de escritorio estándar.

Palabras claves: MDA, APLICACIÓN, MOVIL, UML

1. Estructura

En este apartado se presenta la estructura del presente informe la cual toma como base la propuesta en forma general de la guía de informes de avance/finales, realizando agregados o quitando aquellos títulos que no aplican en la presente temática (en cuyo caso también se indica junto al título en cuestión).

1. Estructura **– Agregado -**
2. Introducción
   - 2.1 Selección del Tema
   - 2.2 Definición del Problema
   - 2.3 Justificación del Estudio
   - Limitaciones **– No aplica / no existieron -**
   - 2.4 Alcances del Trabajo
   - 2.5 Objetivos
   - 2.6 Hipótesis
3. Desarrollo:
   - Material y Métodos **– No aplica -**
   - 3.1. Lugar y Tiempo de la Investigación
     - Descripción del Objeto de Estudio **– No Aplica -**
     - Descripción de Población y Muestra **– No Aplica -**
   - 3.2. Diseño de la Investigación
     - Instrumentos de Recolección y Medición de Datos **– No Aplica -**
     - Confiabilidad y Validez de la Medición **– No Aplica -**
     - Métodos de Análisis Estadísticos **– No Aplica -**
   - 3.3. Etapas Ejecutadas **– Agregado –**
     - 3.3.1. Etapa 1 –Relevamiento **– Agregado –**
     - 3.3.2. Etapa 2 –Modelado **– Agregado –**
     - 3.3.3. Etapa 3 –Refinado del Modelo **– Agregado –**
     - 3.3.4. Etapa 4 – Análisis del XMI **– Agregado –**
     - 3.3.5. Etapa 5 –Construcción de una Herramienta **– Agregado -**
     - 3.3.6. Etapa 6 – Validación y Pruebas **– Agregado -**
     - 3.3.7. Etapa 7 – Utilización de la Herramienta. **– Agregado –**
4. Resultados
   - Discusión **– No Aplica –**
5. Producción Científico-Tecnológica
6. Conclusiones
7. Bibliografía
8. Anexos
   - 8.1. Anexo A –QVT **– Agregado –**
   - 8.2. Anexo B –XSLT **– Agregado –**
   - 8.3. Anexo C-Clase Interoperabilidad **– Agregado –**

2. Introducción

2.1. Selección del Tema

Los sistemas hipermedia más difundidos, son los sitios web, donde el usuario puede visualizar: imágenes, texto, componentes multimedia; además de navegar en el sitio mediante hipertexto. Un caso particular de los sitios web, son aquellos diseñados especialmente para dispositivos móviles. A pesar de la gran evolución de los equipos celulares que cuentan con una alta capacidad de procesamiento, estos siguen teniendo las siguientes limitaciones y necesidades: Pantalla de tamaño reducido; Necesidad de controles simples; Mostrar la información de forma sencilla y directa, sin las complejas distribuciones de pantalla de los sitios web convencionales; Sistema de navegación práctico e intuitivo; Ingreso de texto sencillo. El consorcio W3C (World Wide Web Consortium) presenta una guía de buenas prácticas para el diseño de sitios web móviles [W3C08] donde establece la mejor forma de solucionar los puntos mencionados anteriormente para lograr una web usable desde cualquier dispositivo móvil.

El diseño de sistemas de hipermedia es un área de estudio que data de varios años atrás cuando la web recién comenzaba a popularizarse. Unos de los primeros trabajos en el área es OOHDM (Object Oriented Hypermedia Design Method) [7] que establece cuatro actividades principales en el diseño hipermedia que luego fueron consideradas por trabajos posteriores. Estas actividades son:

- Diseño conceptual: es aquel en el cual se establecen las distintas clases que van a formar el modelo del domino de nuestra aplicación. El enfoque más utilizado es el diagrama de clases como por ejemplo en OOHDM y UWE (UML Based Web Engineering) [KOS08]. Otros trabajos utilizan diagramas de entidad relación como WebML (Web Modeling Language) [CER00] y RMM (Relationship Management Methodology) [ISA95].

- Diseño de la navegación: consiste en construir un modelo que permita identificar los distintos caminos que el usuario podrá seguir al utilizar la aplicación. Este punto es particular de los sistemas de hipermedia donde el usuario dispone de un entorno no lineal donde puede seguir distintos vínculos para navegar por las entidades del sistema por ejemplo el hipertexto en un sitio web. En este aspecto la mayoría de los métodos existentes se basan en la creación de un diagrama de clases particular con la ayuda de estereotipos que complementan las clases navegables, es decir que la navegación se deriva de modelos estructurales. MDHDM (Desarrollo de Hipermedia Dirigido

por Modelos) [PIN08] complementa la navegación mediante un modelo de procesos que permite que la lógica del negocio pueda determinar en cierta forma la navegación del usuario en el sistema dándole más dinamismo.

- Diseño de la interfaz abstracta: que define de una forma independiente de la tecnología la forma en que el usuario verá e interactuará con la aplicación. OOHDM utiliza ADV (Abstract Data Views) [COW95] donde se van agrupando distintos objetos de la interfaz para armar un prototipo de la pantalla. WebML crea una notación gráfica propia para definir las pantallas pero basadas en XML lo que permite fácilmente realizar transformaciones para generar código por ejemplo en HTML.

- Implementación: se trata de la construcción de la aplicación con los modelos definidos. Aquí los distintos métodos tratan de automatizar lo más posible la construcción del software resultante. WebML lo realiza mediante transformaciones XSLT (Extensible Stylesheet Language Transformation) [W3C99]. OOHDM indica el mapeo que debe realizarse en cada uno de los modelos para el desarrollo del software y propone una herramienta que permita diseñar una aplicación utilizando dicho modelo y derivar la aplicación relacionada. MDHDM realiza transformaciones para llegar a código fuente, bajo un entorno Java.

2.2. Definición del Problema

El modelado de aplicaciones es un área muchas veces subestimada por la industria donde no se le da la suficiente importancia y muchas empresas, principalmente pequeñas o medianas, lo consideran una pérdida de tiempo. En otros casos sólo se utiliza en etapas tempranas del desarrollo para hacer una primera definición del problema y obtener los requerimientos. Gran parte de los modelos realizados en estas etapas luego no son actualizados con los cambios que surgen en la etapa de desarrollo haciendo que la documentación del sistema quede obsoleta. El modelado a bajo nivel de una aplicación también es una tarea ardua que agrega costos y tiempos que muchas empresas no están dispuestas a afrontar. Para subsanar todos estos problemas nace la arquitectura dirigida por modelos (MDA) [KLE03], [OMG03] donde los modelos van evolucionando y transformándose hasta llegar a generar el código fuente o parte del mismo en forma automática. Si bien actualmente existen herramientas basadas en esta arquitectura que generan código fuente, ninguna de ellas permite generar una aplicación completa 100% funcional, e incluso el esfuerzo

necesario para realizar un modelado detallado que permita lograr dicha meta es demasiado elevado.

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

2.3. Justificación del Estudio

Si bien existen metodologías de modelado ninguna de ellas logra generar en forma completa una aplicación completamente funcional sin necesidad de modificar luego el código fuente. Este proyecto tiene gran importancia en el área de Ingeniería de Software en el ámbito de MDA (Arquitectura Dirigida por Modelos) ya que está metodología plantea las transformaciones necesarias a partir de modelos para lograr obtener líneas de código fuente.

2.4. Alcances del Trabajo

Desarrollar una metodología de modelado que contenga toda la información necesaria para poder generar tanto las bases de datos como el código fuente funcional de una aplicación web móvil principalmente centrada en el manejo de datos. La metodología estará basa en UML permitiendo extendiéndolo de forma conservativa para poder ser empleada con cualquier herramienta de modelado actual.

2.5. Objetivos

- Definir una metodología de modelado basada en una extensión conservativa de UML que pueda ser realizada con cualquier herramienta existente.
- Construir una herramienta que permita tanto la transformación entre diagramas como la de diagramas a código fuente.
- Generar aplicaciones funcionales a partir de diagramas de UML.
- Construir aplicaciones móviles tanto para celulares básicos (en XHTML 1.1 Basic) como para celulares avanzados (HTML 5).

2.6. Hipótesis

Es posible definir una metodología de modelado que permita generar aplicaciones en forma totalmente automática a partir de diagramas basados en UML (Lenguaje Unificado de Modelado) sin necesidad de tener luego que modificar o completar el código fuente generado.

3. Desarrollo

3.1. Lugar y Tiempo de la Investigación

Las tareas se realizan dentro del laboratorio designado para este proyecto. El DIIT cuenta con un laboratorio de investigación en el cual se encuentra el grupo GIDFIS (Grupo de Investigación Desarrollo y Formación en Innovación de Software). Los tiempos de las tareas se llevaron a cabo en base al GANTT diseñado previamente en el momento de la presentación del presente proyecto, se muestra a continuación las actividades planificadas para el primer y segundo año que son las correspondientes al presente informe final.

El único cambio efectuado está vinculado con las siglas de las personas asignadas por tarea ya que se dio de baja por motivos personales a un alumno que forma parte del equipo: Alan Gabriel Vivona (AGV) y se incorporaron otros 2 alumnos al proyecto: Javier Gastón Cescon (JGC), Gaspar Acevedo Zain (GAZ); así también se incorporó con horas ADHonorem a prueba a un docente de la universidad Roberto Mariano Ariel Bucher (RMAB). Esto hizo que deban asignarse tareas a los nuevos miembros y reasignarse las tareas de AGV a otros miembros. No hay cambios en cuento a tareas a efectuar ni a la duración de las mismas respetándose el cronograma original.

Actividades / Responsables 1er Año	Mes 1	Mes 2	Mes 3	Mes 4	Mes 5	Mes 6	Mes 7	Mes 8	Mes 9	Mes 10	Mes 11	Mes 12
ETAPA 1 – Relevamiento	**X**	**X**	**X**	**X**								
1- Estado del Arte RAR-DAG-CGA-AGC a) Síntesis de Material de lectura (MDA, XMI, etc.) b) Síntesis publicaciones científicas en el área c) Resumen de investigaciones académicas de otros grupos	X	X	X	X								
ETAPA 2- Modelado				**X**	**X**	**X**	**X**	**X**				
2- Generación de un profile (perfil de UML) AGV, FEV, PMV, VMF a) Generación de Estereotipos b) Análisis de las clases de las que derivan esos estereotipos c) Creación del MOF (Meta Object Facility)				X	X	X						
3- Modelado de un caso de prueba RAR, VMF, AGV a) Selección de Software para el modelado b) Diseño del primer caso de ejemplo c) Modelado del ejemplo con las distintas herramientas CB, MC, PMV, VF, RMAB							X	X				
ETAPÀ 3 – Refinado del Modelo								**X**	**X**	**X**		
4- Análisis de nuevas necesidades AGC, DAG, CGA, AGV, CGA, RAR a) Selección de nuevos casos de ejemplo con distintos grados de complejidad b) Modelado de los casos de ejemplo c) Reformas al profile en base a la evidencia de nuevas necesidades												
ETAPA 4 – Análisis del XMI										**X**	**X**	**X**
VMF, PMV, AGV, CGA, FEV, GAZ 5- Analisis del XMI a) Analizar los versionados de XMI a los que exporta cada uno de los Software b) Seleccionar el versionado de trabajo c) Exportar cada modelo realizado en las distintas herramientas en un XMI d) Analizar la correspondencia (mapeo) entre los elementos del diagrama a cada uno de los elementos del XMI												
6- Informe de Avance RAR, DAG, AGC												**X**

Actividades / Responsables 2do Año	Mes 1	Mes 2	Mes 3	Mes 4	Mes 5	Mes 6	Mes 7	Mes 8	Mes 9	Mes 10	Mes 11	Mes 12
ETAPA 5 – Construcción de una herramienta	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**				
7- Definición del BNF (Backups Naur Form) PMV, CGA, AGC, FEV,AGV,RAR	X	X										
8- Construcción del Parser DAG, PMG, AGV		X	X	X								
9- Transformación entre modelos DAG, PMV, CGA, VMF				X	X							
10- Definición de la estructura de los Templates DAG, PMV, CGA, VMF					X	X						
11- Armado de un template de ejemplo RAR, DAG, CGA, VMF						X	X					
12- Generación a código fuente y generación de la base de datos PMV, VMF, AGV						X	X	X				
ETAPA 6 – Validación y Pruebas									**X**	**X**		
13- Testeo de la herramienta construida RAR, DAG, AGC, AGV a) Validación de los resultados de la herramienta b) Pruebas c) Depuración de posibles errores												
ETAPA 7 – Utilización de la herramienta										**X**	**X**	**X**
14- Se utiliza la herramienta para los casos planteados en la etapa 3 DAG, RAR, IBM, EJM,MAC										X	X	
15- Planteo de nuevos casos posibles PMV,CGA, AGC, FEV,AGV,RAR											X	X
16- INFORME FINAL DEL PROYECTO RAR, DAG, AGC												**X**

3.2. Diseño de la Investigación

Como puede verse en el GANTT presentado en el apartado anterior las tareas del proyecto se encuentran divididas por Etapas. En la figura 1 se muestran las 7 etapas a realizar a lo largo del proyecto, las 4 primeras etapas se corresponden con el primer año del proyecto, las siguientes 3 etapas corresponden al segundo año del proyecto. Todas representadas en el Gantt.

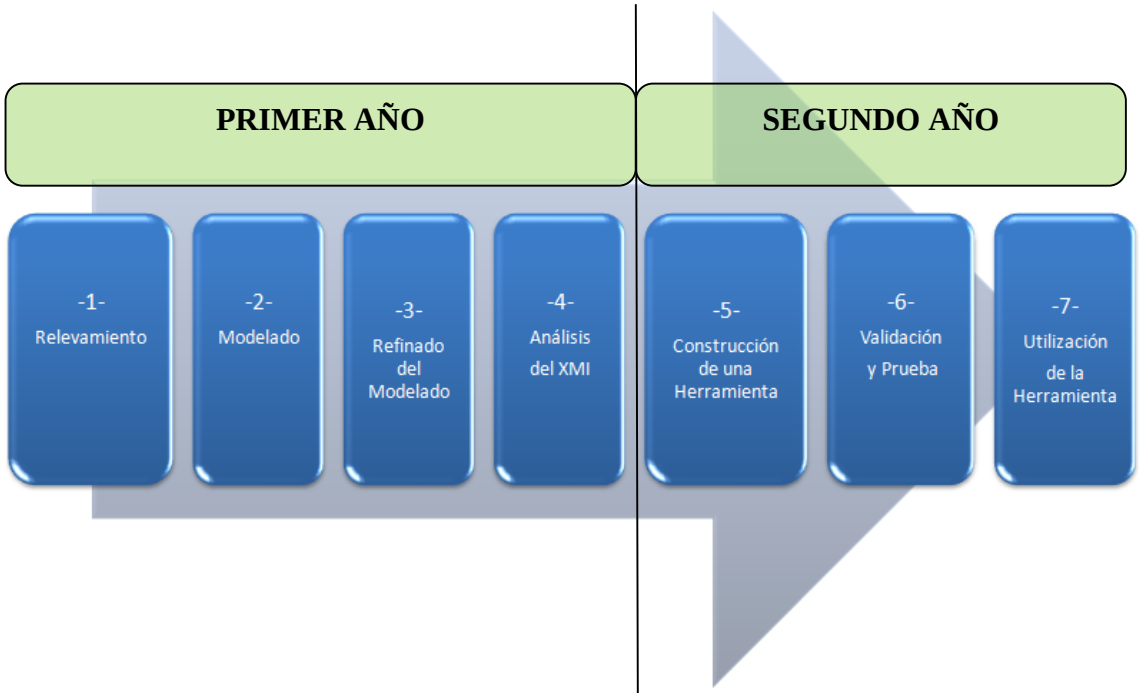


Figura 1. Etapas del Proyecto de Investigación.

3.3. Etapas Ejecutadas

En el presente trabajo se presentan el desarrollo de las distintas actividades que corresponden a cada una de las etapas ejecutadas.

3.3.1. Etapa 1 – Relevamiento

En ésta primera etapa se puede observar la exposición del estado del Arte.

3.3.1.1. Tarea1 – Estado del Arte

3.3.1.1.1. Características de MDA

La Arquitectura Dirigida por Modelos (Model Driven Arquitecture, MDA [OMG03]) es una de iniciativas de la OMG. La esencia del enfoque MDA es que los modelos son la base del desarrollo del software, los cuáles deberían ser buenos, sólidos, consistentes y coherentes.

Dirigido por modelos, significa que los mismos proporcionan un medio para la comprensión y la dirección del curso del diseño, construcción, implementación, operación, mantenimiento y modificación.

MDA construida bajo la supervisión de la OMG:

- Especifica como los modelos definidos en un lenguaje, pueden ser transformados en modelos en otros lenguajes.
- Plantea a partir de metamodelos, la definición, intercambio y uso de los modelos. Los cuales se utilizarán para la simulación y testeo de los mismos y la generación de código.

MDA proporcionará:

- Una plataforma de modelos comunes que serán independientes de servicios generalizados.
- Asignaciones para los modelos de transformación: se basan en esos servicios generalizados (PIM), los cuales se transforman a modelos específicos de plataforma, usando los servicios proporcionados por una plataforma particular.

La figura 2 resume los beneficios de ésta arquitectura.

Figura 2. Beneficios de la Arquitectura.

Se intenta conseguir para los sistemas a desarrollar, la portabilidad, interoperabilidad y reusabilidad. El proceso MDA está dividido en tres pasos (ver figura 3).

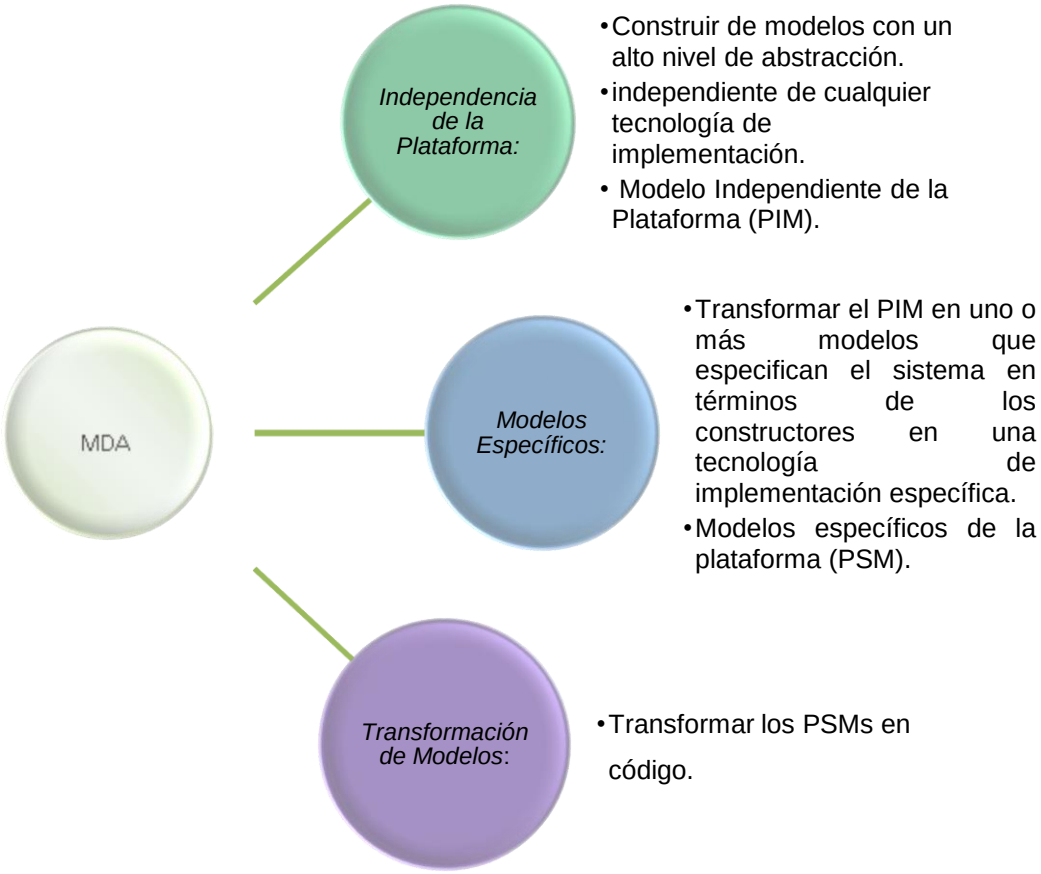


Figura 3. MDA en tres pasos.

Dentro del proceso MDA, el foco está puesto en producir un modelo del sistema de alto nivel.

Figura 4. Proceso MDA

Modelo Independiente de la Plataforma (PIM) Platform Independent Model

Un modelo independiente de la plataforma es una vista de un sistema independiente desde el punto de vista de la plataforma. Un PIM exhibe un grado específico de independencia de la plataforma a fin de ser adecuado para su uso con un número de diferentes plataformas de tipo similar.

Una técnica muy común para el logro de la independencia de plataforma es apuntar a un modelo de sistema de una máquina virtual de tecnológicamente neutral. Una máquina virtual

se define como un conjunto de piezas y servicios (comunicaciones, programación, asignación de nombres, etc), que se definen de forma independiente de cualquier plataforma específica.

Modelo Específico de la Plataforma (PSM) Platform Specific Model
Un modelo específico de la plataforma es una vista de un sistema desde el punto de vista específico de la plataforma.

Un PSM combina las especificaciones de la PIM con los detalles que especifican la forma en que el sistema utiliza un determinado tipo de plataforma.

TRANSFORMACIÓN DE MODELOS
La transformación de modelos es el proceso de convertir un modelo en otro modelo de un mismo sistema.
La figura 5 ilustra el patrón MDA por el cual un PIM es transformado en un PSM.

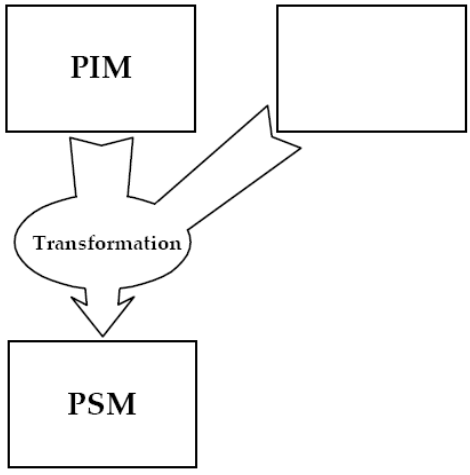


Figura 5. Patrón MDA. (MDA Guide Version 1.0.1)

El paso más complejo es cuando un PIM es transformado en un PSM (ver figura 6).

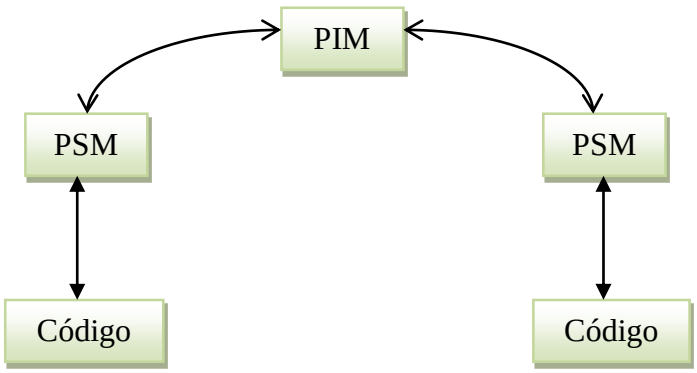


Figura 6. Transformaciones.

Tipos de Transfor

- **Transformación manual:** Al hacer la transformación de PIM a PSM, se deben tomar decisiones de diseño. Estas decisiones se pueden hacer durante el proceso de desarrollo de un diseño que cumple con los requisitos de ingeniería en la aplicación. Este enfoque es muy útil, debido a que estas decisiones son consideradas y tomadas en el contexto de un diseño de implementación especifico.
  Este proceso de transformación manual tiene lo bueno del trabajo de diseño de software y el enfoque MDA agrega valor de dos maneras:
  - Distingue explícitamente entre un modelo independiente de la plataforma y de uno específico de la plataforma.
  - Registra la transformación.

- **Transformación automática:** Hay contextos en los que un PIM ofrece toda la información necesaria para la aplicación, y no hay necesidad de añadir marcas o utilizar datos de profiles adicionales, con el fin de ser capaz de generar el código. Uno de ellos es el basado en componentes de desarrollo maduros, donde el middleware proporciona un conjunto, completo de servicios, y en el que las decisiones arquitectónicas necesarias se realizan una vez para una serie de proyectos, todas las construcciones de sistemas similares. Estas decisiones se implementan en las herramientas, los procesos de desarrollo, plantillas, bibliotecas de programas, y generadores de código.

En tal contexto, es posible para un desarrollo de una aplicación construir un PIM que seas completo en cuanto a la clasificación, estructura, invariantes, y pre- y post-condiciones.
El desarrollador puede especificar el comportamiento requerido directamente en el modelo, utilizando un lenguaje de acción. Esto hace que el PIM sea completo computacionalmente, es decir, el PIM contiene toda la información necesaria para producir código de programa informático.

En este contexto, el desarrollador no tiene que ver un PSM, ni es necesario añadir información adicional al PIM, además de que disponer la herramienta de transformación. La herramienta interpreta el modelo directamente o transforma el modelo directamente al código de programa.

Un determinado PIM, en una tienda de desarrollo de componentes maduros, con un estilo arquitectónico establecido y con las decisiones de ingeniería específicas de la plataforma ya realizadas y que han sido reutilizados, pueden ser usados para generar código (por ejemplo:

componentes en su forma de código) no sólo componentes de plataformas J2EE, sino también a algunas de las otras plataformas de aplicaciones de servicios.

Para su reutilización se ha de preparar:

1. Un modelo de la arquitectura
2. Los detalles dentro de ese modelo, como un sistema de tipo PIM, que se pueden asignar automáticamente a las distintas plataformas de destino.
3. El soporte de la herramienta necesaria para suministrar el modelo para los desarrolladores en forma de profiles, controles de conformidad de modelo, enlaces a un IDE, el apoyo a los procesos, y así sucesivamente.
4. Una asignación para cada plataforma de destino.

Con este tipo de entorno de desarrollo, el desarrollador de aplicaciones necesita desarrollar sólo un PIM, y el código puede ser generado directamente para ese PIM.

Cómo funciona MDA

“En el nivel más alto se encuentra el modelo independiente de la Computación (CIM). Este modelo CIM contiene información acerca de los procesos de negocio y si se van a implementar o no.

En el segundo nivel se encuentra el modelo independiente de la plataforma (PIM). Es el modelo en el que comienza la mayor parte de las herramientas de la MDA. Se usa un diagrama de clases, a veces llamado un modelo de dominio. Al ser independiente de la plataforma, un PIM puede satisfacer múltiples plataformas y puede transformarse en uno o más modelos específicos de plataforma (PSM). Para cada tecnología se necesita un PSM. Por ejemplo, si se tiene una aplicación de muchos niveles se realizará un PSM para cada nivel. Podría ser, uno para su base de datos, una para los EJB y otro para las JSP. Esto se muestra en la figura 7.

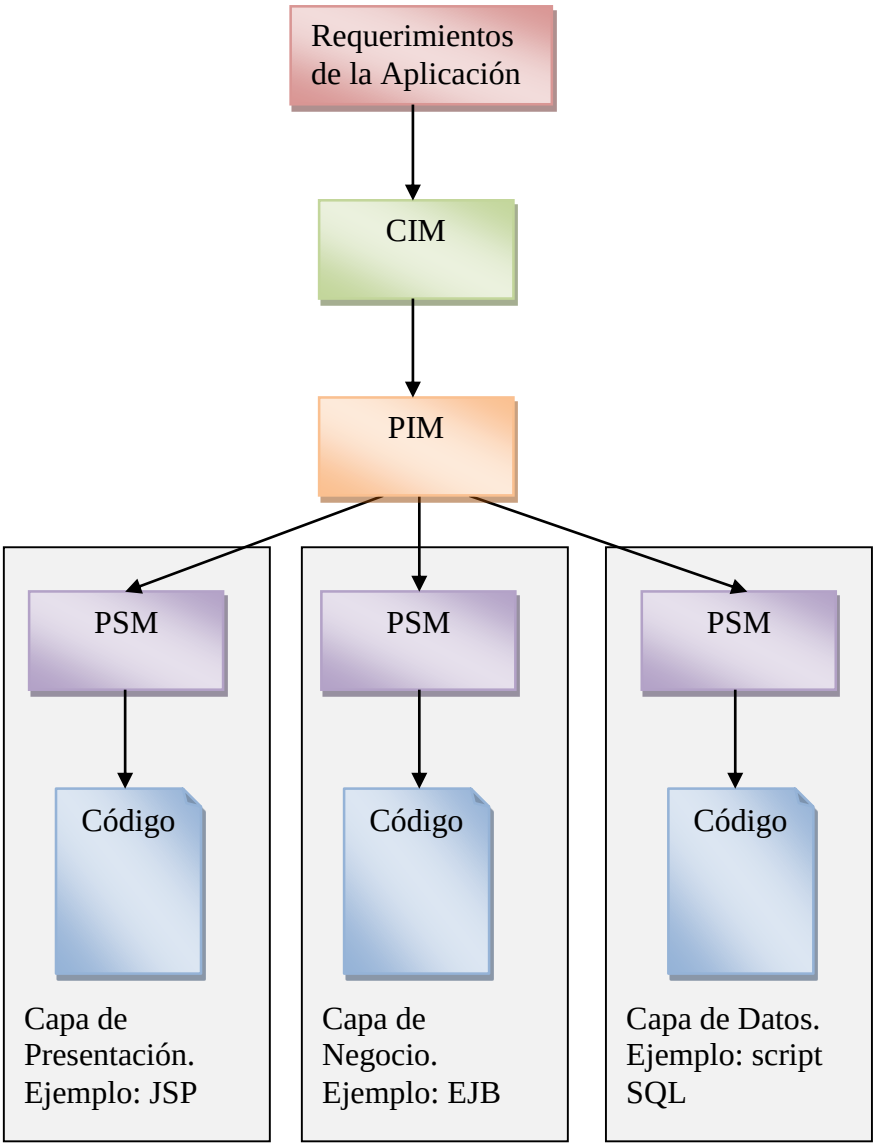


Figura 7. Desarrollo en tres capas con MDA. (Ejemplo extraído: www.technology.amis.nl)

3.3.1.1.2. Metodologías.

3.3.1.1.2.1. HDM (Hypertext Design Model) ó Modelo de Diseño Hipermedia.

- **Características**

  El HDM surgió con el objetivo de crear un modelo que fuera de utilidad para realizar el diseño de una aplicación de hipertexto.

  El modelo HDM divide el proceso de diseño de una aplicación de hipertexto 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, y el authoring-in-the-small, que se refiere al desarrollo del contenido de los nodos.

  El modelo se centra en la primera parte, que es la que se enfoca en la estructura.

HDM utiliza el Modelo Relacional para representar las aplicaciones hipertextuales.

- **Etapas de diseño**

  En HDM se distinguen dos etapas en el diseño de aplicaciones: diseño a gran escala (design in the large), y el diseño a pequeña escala (design in the small).

  El diseño a gran escala se refiere al diseño global y a los aspectos estructurales de la aplicación hipermedia, en otras palabras, trata de la dentición de las relaciones conceptuales entre los nodos de la aplicación hipermedia.

  El diseño a pequeña escala se refiere al desarrollo del contenido de los nodos de hipermedia, y está relacionado con la implementación de estos, pues trata, de solucionar problemas como la 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 diseño a pequeña escala prácticamente no es abordado en el método y se limita solo a la asignación de contenido a los nodos. 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.

- **Notación**

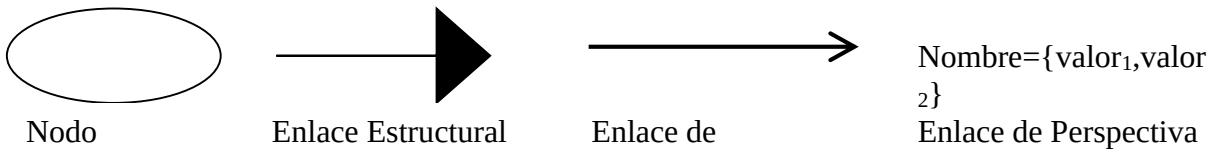


Figura 8. Notación en HDM.

- **Estructura**

✓ **Nodos**

La información esta agrupada en nodos, representados mediante óvalos e identificados por un nombre.

HDM distingue los siguientes tipos de nodos:

* Una unidad es un nodo que no se compone por ningún otro y es una hoja en la jerarquıa.
* Un componente esta integrado por un conjunto de unidades o componentes, estos tienen una relacion de pertenencia con el componente superior, es decir, los componentes forman jerarquıas.
* Una entidad es una estructura de informacion que describe un objeto abstracto o real, que es parte del dominio de aplicacion.
* Una entidad define un tipo, y cada entidad del mismo tipo comparte la misma estructura y conexiones con las demas, y es un componente raız.

La figura 9 muestra la jerarquía de los nodos.

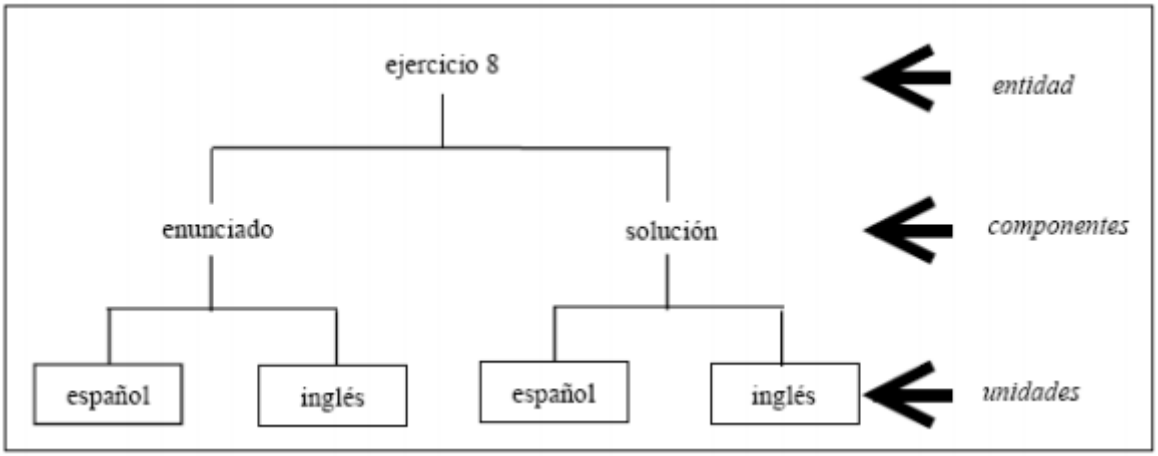


Figura 9. Jerarquía de Nodos en HDM.

✓ **Unidades**

Una unidad es un conjunto de partes atómicas de información que se muestran conjuntamente, como una unidad.

Una unidad corresponde a un componente asociado con una perspectiva específica. Una unidad se caracteriza por un nombre (su identificador) y un cuerpo. Los cuerpos de las unidades son el lugar donde queda almacenada la información.

✓ **Componentes**

Un componente es una abstracción de un conjunto de unidades, que son los contenedores de la información.

Un componente sólo puede existir como parte de una entidad, y no tiene sentido por sí mismo, es decir, no es autónomo. Por tal razón en el modelo HDM no se permite que unas entidades sean componentes de otras entidades.

✓ **Entidades y tipos de entidades**

Una entidad es una estructura de información que representa algún objeto del mundo real en el dominio de la aplicación.

Las entidades se agrupan lógicamente en lo que llamamos tipos de entidades, que corresponden a las clases de objetos del mundo real.

- Entidades: Representan objetos, físicos o conceptuales, que tienen significado y existencia propia dentro del dominio de aplicación.
- Tipos de entidades: Permiten agrupar entidades.

- **Navegación**

✓ **Enlaces**

Los nodos se relacionan mediante enlaces, que pueden ser, también, de tres tipos:

1. **Enlaces estructurales:**
   - Representan relaciones jerárquicas de la estructura.
   - Sirven para conectar componentes que pertenecen a la misma entidad, es decir, conectan las entidades, componentes y unidades de una jerarquía.
   - Este tipo de enlaces se identifican por una flecha triangular.

2. **Enlaces de aplicación:**
   - Representan relaciones de dominio entre entidades o sus componentes.
   - Sirven para conectar nodos de distintas jerarquías.
   - Los enlaces de aplicación se estructuran en tipos, llamados tipos de enlaces de aplicación, o más brevemente, tipos de

enlaces. El tipo viene definido por un nombre, los tipos de entidades origen y destino, y por si es simétrico o no. Los enlaces de aplicación pueden ser de dos tipos diferentes:

- De esquema: pertenece a un determinado tipo de enlaces.
- Genéricos: no pertenece a ningún tipo de enlace. Son independientes de la estructura, y por tanto menos estandarizados.

- Este tipo de enlace se representa por una flecha normal.

3. **Enlaces de perspectiva o de componentes:**
   - Un enlace de perspectiva corresponde a una vista de la aplicación.
   - Interconectan unidades dentro del mismo componente.
   - Los enlaces de componente pueden ser de dos formas:
     - enlaces de índice
     - enlaces de visita guiada

✓ **Estructuras de Acceso**

Los otros elementos de navegación son las estructuras de acceso o colecciones.

Una colección es un conjunto de enlaces que apuntan a nodos hipermedia.

Las colecciones pueden ser de dos tipos: índices y visitas guiadas.

- Un índice permite navegar directamente a los nodos que pertenecen a la colección.
- Una visita guiada muestra la colección de objetos a través de un recorrido secuencial hacia adelante y atrás.

Las estructuras de acceso son componentes y no tienen un símbolo para especificarlas, solo se indican, adicionalmente, en el nombre del nodo como etiqueta.

- **Perspectivas**

En HDM la noción de tener diferentes presentaciones del mismo contenido se representa mediante el concepto de perspectiva.

Las perspectivas son sólo una construcción para organizar este tipo de información.

HDM deja a la libertad del usuario cuándo y cómo debe utilizar perspectivas.

La figura muestra un modelo HDM de una librería electrónica, la cual esta conformada por libros clasificados en colecciones de alguna editorial.

Editorial, libro y autor son las entidades o componentes del nivel mas alto. El nodo capıtulo es un componente, pero no una entidad. La jerarquıa correspondiente a editorial se descompone en colecciones (de libros), la de libro en capıtulos, y estos a la vez en secciones. La entidad autor forma una jerarquıa con los nodos ındice de libros y biografıa, los cuales son unidades. Los enlaces que conforman la jerarquıa son estructurales, y los que interconectan las jerarquıas son enlaces de aplicacion. Los libros tienen la perspectiva idioma que puede ser ingles o español, de modo que los nodos capıtulo y seccion, se pueden agrupar mediante esa perspectiva.

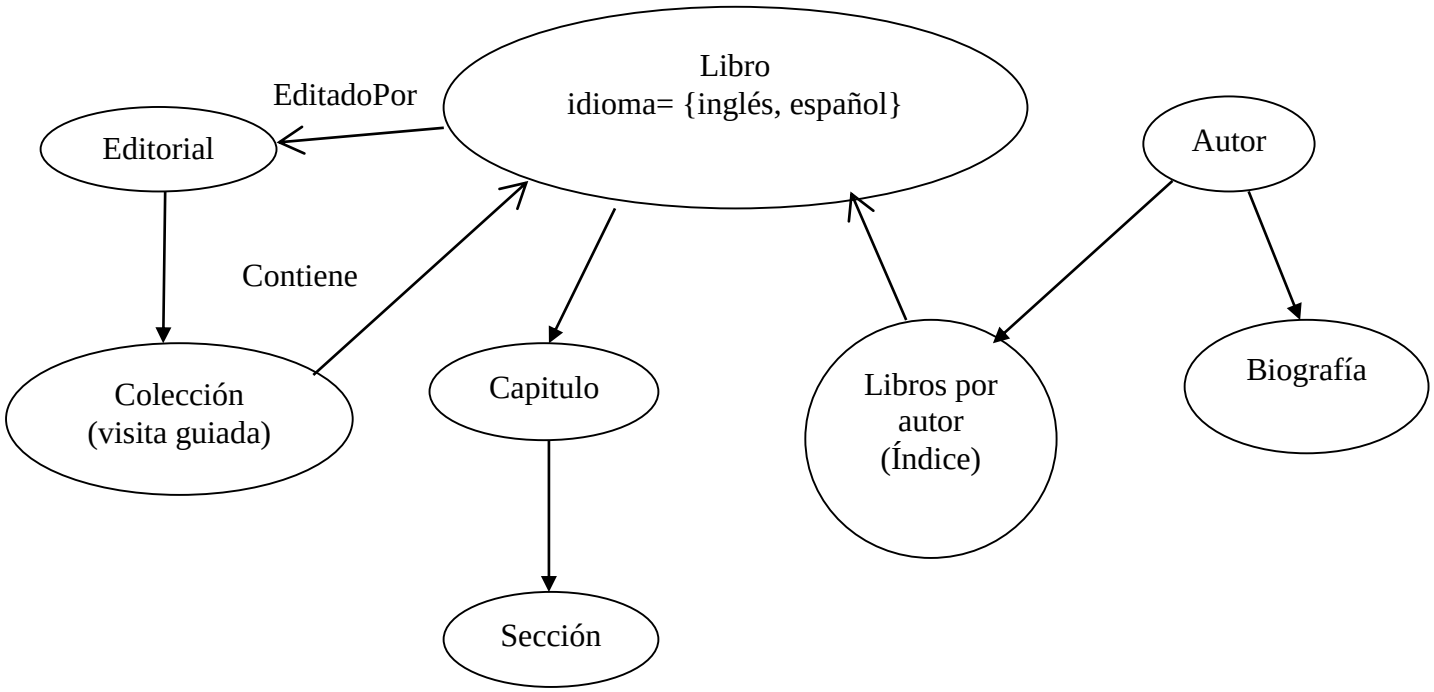


Figura 10. Modelo HDM de una librería electrónica.

3.3.1.1.2.2. OOHDM

OODHM[1] es una Metodología de Diseño de Hipermedia Orientada a Objetos (Object Oriented Hypermedia Design Methodology). [LAM11]

Fue concebida con la necesidad de solucionar problemas encontrados en la etapa de diseño de aplicaciones web dinámicas con un grado de complejidad alto, o donde se desea combinar patrones de navegación complejos y sofisticados. [LAM11]

Esta metodología, diseñada por D. Schwabe, G. Rossi, y S. D. J. Barbosa, es una extensión del Modelo de Diseño de Hipermedia, (HDM, Hypertext Design Model) [SCHNN]. Utilizada en el diseño de aplicaciones web, pudiendo contener presentaciones multimedia y galerías interactivas entre otras. [LAM11]

La metodología describe un proceso de cuatro (4) actividades principales o etapas [LAM11]:

- Diseño conceptual
- Diseño de navegación
- Diseño de interfaz abstracta
- Implementación (en ésta última se ve reflejado como se implementan los diseños: conceptual, de navegación y de interfaz abstracta en una aplicación web).

A continuación se describe brevemente cada una de ellas:

- **Diseño Conceptual:** Utiliza principios orientado a objetos pudiéndose utilizar una notación similar a UML, con múltiples atributos de valores e instrucciones explícitas en las relaciones [SCH98]. OOHDM fue concebida para que en su utilización no se exiga ningún método en particular para producir el esquema de clases. Se deberá construir el Modelo de Dominio conceptual de la aplicación lo más objetivamente posible. (Esto significa que deberá ser lo más neutral posible). Productos: Clases, relaciones y subsistemas. (Clases conceptuales con agregación y jerarquías de generalización / especialización.)

- **Diseño Navegacional:** Una característica particular de OOHDM es que la aplicación es considerada como una vista de navegación del modelo conceptual. Esta es una de las principales diferencias de ésta metodología, reconocer los ítems u objetos que el usuario navega que son construcciones de uno o más

[1] Object Oriented Hypermedia Design Methodology.

objetos conceptuales. Existe una fuerte tendencia a poner el énfasis en la navegación. [SCH98a]

Un modelo de navegación está construido como una vista sobre un modelo conceptual, permitiendo así la construcción de diferentes modelos de acuerdo a diferentes perfiles de los usuarios. Cada modelo de navegación proporciona una visión subjetiva del Modelo conceptual.. [SCH98a].

El modelo conceptual obtenido del paso anterior es la materia prima para realizar este modelo. Las relaciones dan orígen a los enlaces y las clases conceptuales a los nodos que representan ventanas lógicas o vistas (views).

Los nodos son contenedores de información básica en aplicaciones multimedias. Se podría decir que son construcciones que pueden representar ser una idea, noción ó contexto, desde el punto de vista de un observador. [SCH98a]

Los atributos de nodo se definen usando un lenguaje de consulta. Esto permite que un nodo se defina mediante el acceso a los atributos de diferentes clases que están relacionadas en el modelo conceptual (o mediante la combinación de ellos). Esta particularidad permite enfocarse a una cierta clase de usuarios y sus tareas respectivas. Resalta e individualiza a cada usuario con las responsabilidades o habilidades que puede realizar en el dominio de la aplicación. [SCH98a]

Existe además la idea de contextos navegacionales donde interactuan un conjunto de nodos y enlaces para los cuales se deberá describir la estructura navegacional.

La estructura navegacional, (estructura de navegación de la aplicación hipermedia) se representa en un esquema donde se especifican clases de navegación. Existen un conjunto de tipos predefinidos de clases navegacionales: nodos, enlaces y estructuras de acceso, los que son habituales y característicos de las aplicaciones hipermedia. Las estructuras de acceso, como lo son los indices, visitas guiadas, y otras, representan formas posibles de acceso a los nodos que actúan como índices o diccionarios y son útiles ayudando a los usuarios finales a encontrar la información deseada. [SCH98a]

El origen y carácter principal de estructurar un esquema de navegación es concebir la noción de un contexto navegacional.

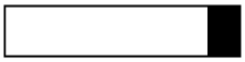
Se llama Contexto de navegación (ó navegacional) al conjunto de clases de navegación (nodos, enlaces), clases de contextos y otros contextos navegacionales (contextos anidados), los que pueden definirse por comprensión

(por su significado) o por extensión (enumenrando los elementos que lo compone). [LAMNN]

Es posible adaptar los objetos navegacionales para cada contexto. Cuando se especifican las clases navegacionales se definen las correspondencias.
Hay seis formas diferentes de definir contextos [SCH98a], ver tabla 1.

Tabla 1. Definición de contextos.

Definición de Contexto	Definición	Gráfica
1. Derivado de Clase Simple.	Todos los objetos de una clase que satisfacen alguna propiedad, por ejemplo, "profesores con grado asociado iguales". Una variante de este tipo es el contexto de consulta basado en el que se define la propiedad por el usuario en tiempo de navegación.	
2. Derivado de Grupo de clases.	Un conjunto de contextos de clase simples derivadas, donde se parametriza la propiedad que define a cada contexto, por ejemplo, "Profesores por rango" (rango puede variar).	
3 - Derivado de Enlace Simple	Todos los objetos están relacionados con un determinado objeto dado, por ejemplo, "los cursos impartidos por el profesor Smith".	 *Igual que (1)*
4 –Grupo Derivado de enlace.	Un conjunto de contextos derivados del enlace, cada uno de los cuales se obtiene variando el elemento de origen del enlace; por ejemplo,	 *Igual que (2)*

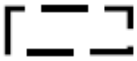
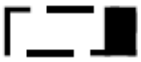
	"*Cursos impartidos por el profesor*" (varía profesor).	
5 - Arbitraria (ó Enumerado)	Es un conjunto enumerado, por ejemplo, una "*visita guiada*".	 *Igual que (1)*
6 – Dinámico	Es un conjunto donde los elementos cambian durante la navegación; por ejemplo, la "*historia o historial*" (se va construyendo a medida que se navega), otro ejemplo es un "*shopping ó carrito de compra* ", el usuario construye durante la navegación a través de objetos que va seleccionando para la compra (por ejemplo, libros) en otros contextos.	

Se pueden precisar entonces dos niveles de abstracción o dos diferentes vistas definidas en éste modelo:

- La especificación de las ***Clases navegacionales***, cuando se definen las correspondencias de las clases del modelo de dominio a nodos y de las relaciones a los enlaces.
- La especificación de la ***Navegación:*** la cuál está expresada exclusivamente sobre los objetos navegacionales. Contextos Navegacionales.

Asociados a los contextos, hay estructuras de acceso (índices). Ellos son denotados gráficamente tal como se muestra en la tabla 2.

Tabla 2. Estructuras de Acceso y gráficas asociadas. [SC98a]

Estructuras de Acceso (índices)	Gráficas
Índices:	
Índices dinámicos:	
Índices con ordenamientos múltiples:	

En la etapa de diseño de la estructura de navegación de una aplicación web, se deberá definir:

- Los Objetos que se usen para navegar con sus respectivos atributos y las relaciones entre ellos, (en la definición de nodos y enlaces).
- Las estructuras de composición que existen entre los objetos de navegación y como se relacionan.
- La estructura de navegación que se obtiene como resultado.
- Los contextos que los usuarios navegará.
- Si la navegación de objetos será diferente según el contexto visitado. Debiendo quedar claramente expresadas estas diferencias.
- Las conexiones y estructuras de acceso que existen entre los objetos que se navegarán (enlaces, caminos, índices, guiada-tours, etc).
- El efecto de la navegación, o sea, como será la navegación cuando el usuario salta de un objeto a otro (navegación entre objeto fuente y destino y objetos relacionados).

- **Diseño de Interfaces Abstractas:** El diseño de la interfaz interactiva en aplicaciones y en particular si son web, es un aspecto crítico. [SILNN].

  Al tener separados la navegación y el diseño de interfaz permite tener una independecia con lo cuál se logra un mejor compresión de la estructura completa de la aplicación. Permite construir diferentes interfaces para el mismo modelo navegacional.

  Se especificarán los aspectos de la interfaz, lo que significa:

  - Describir como aparecerán los objetos de intefaz asociados a los objetos de navegación anteriormente definidos.
  - Cuales objetos de inteface serán activados en la navegación y otras funcionaliddes de la aplicación.
  - Que transformaciones de interfaz deberán realizarse y en que momento.

- **Implementación:** El diseñador hará realidad el diseño, lo implemntará. Es decir, es la puesta en marcha de la aplicación, haciendose corresponder los objetos de interfaz con los objetos de implementación. [SCH99].

  Se puede ver claramente como se relaciona cada elemento de la interfaz de la aplicación con la vista abstracta de datos.

Breviario

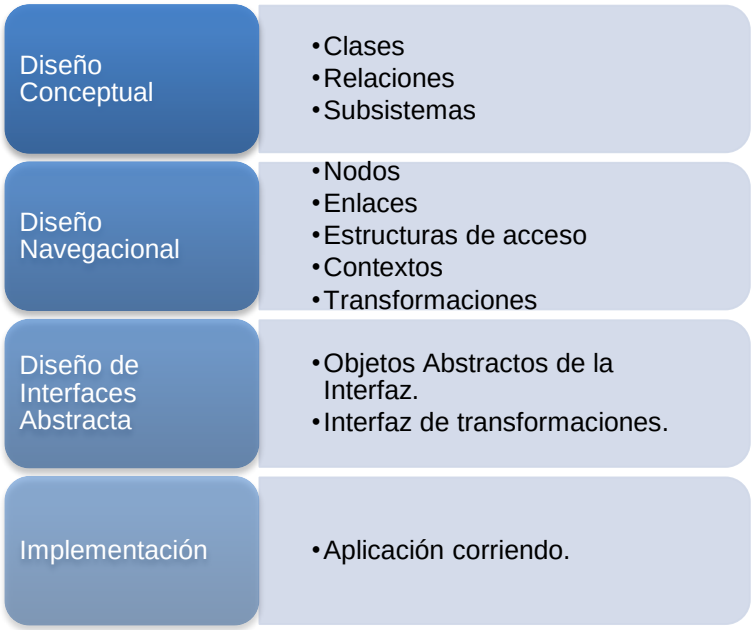


Figura 11. Etapas de OOHDM.

Cada actividad se realiza de forma iterativa e incremental, en cada una de ellas, se realizan un conjunto de modelos con una visión orientada a objetos, donde se describen las características particulares del diseño; el cuál va creciendo a medida que se suceden las iteraciones.

El objetivo de la propuesta es conseguir un marco de desarrollo para los procesos de diseño donde sus productos sean modulares y reutilizables, poniendo énfasis y centrándose en el diseño.
El diseño de la interfaz puede implementarse en lenguajes no orientados a objetos y en diferentes entornos, dando la posibilidad de utilizarse de manera independiente en ambientes híbridos o que no sea puramente orientado a objetos. [SCH98a]

Las características del enfoque OOHDM son [SCH98b]

- Los objetos de navegación son vistas, en el sentido de base de datos, de objetos conceptuales;
- Se modelan abstracciones propias de la navegación, y se definen que objetos o acciones puede realizar cada usuario en particular (contextos de navegación).
- Se separan las características de la interfaz de las particularidades propias de la navegación.

- Se identifican explícitamente las decisiones de diseño que se realizan en tiempo de implementación.

La metodología OOHDM propone los elementos mostrados en la tabla 3 (extraída de "Developing Hypermedia Applications using OOHDM" - Daniel Schwabe y Gustavo Rossi).

Tabla 3. Actividades, Productos y Consideraciones de Diseño. [SCH98a]

Actividades	Productos	Formalismos	Mecanismos	Consideraciones de Diseño (Objetivos)
Diseño Conceptual	• Clases • Subsistemas • relaciones y • posibles atributos.	• Modelar con construcciones Orientadas a Objetos.	• Clasificación • Agregación • Generalización y • Especialización	• Modelar la semántica del Dominio de la Aplicación.
Diseño Navegacion	• Nodos • enlaces • estructuras de acceso • contextos navegacionales • transformaciones navegacionales.	• Paradigma Orientado a Objetos. • Hace énfasis en el estado de los objetos gráficos y clases de contexto.	• Clasificación • Agregación • Generalización y • Especialización	• Tener en cuenta, perfil de usuario y tarea. • Enfasis en aspectos congnitivos. • Construir la estructura de navegación.
Diseño Interface Abstracta	• Objetos abstractos de la interfaz • respuestas de acontecimientos externos • interfaz de transformaciones	• Vistas de datos abstractos. • Diagramas de configuración. • Gráficas ADV (Vista de Datos Abstracta [CO 95])	• Realizar correspondencia entre la navegación y los objetos percibidos • Diseño navegacional → Diseño conceptual	• Modelo de Objetos. • Describir interfaz para los objetos de navegación.
Implemen-tación	• Aplicación corriendo.	• Respaldar con el entorno de destino.	• Previstos en el entorno de destino.	• Rendimiento. • Integridad.

- **Relaciones entre los modelos**

  En la figura 12 (extraída de "An Object Oriented Approach to Web-Based Application Design". - Daniel Schwabe and Gustavo Rossi) se muestran las

relaciones entre los esquemas conceptual, navegacional y los objetos de interfaz en OOHDM.

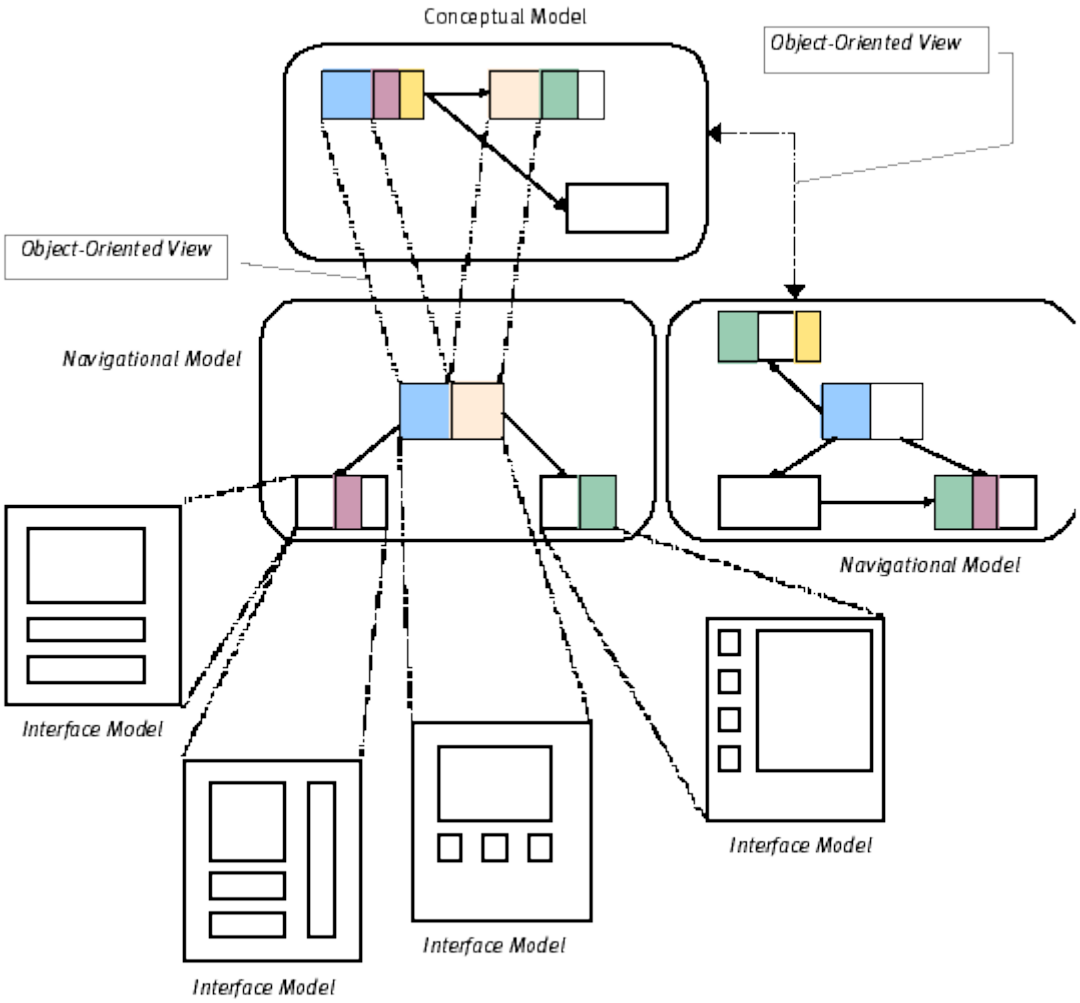


Figura 12. Relaciones entre los Modelos. [SCH98b]

Si se utiliza en el diseño un ambiente CASE para describir los modelos: conceptuales, de navegación e interfaz usando OOHDM, puede proporcionar documentación automatizada sobre dichos modelos. Además puede generar plantillas de aplicación para diferentes entornos.

A cotinuación se presenta a modo de ejemplo el modelado de una revista online, este ejemplo fue extraído de [SCH98b].

✓ **Modelo Conceptual.**

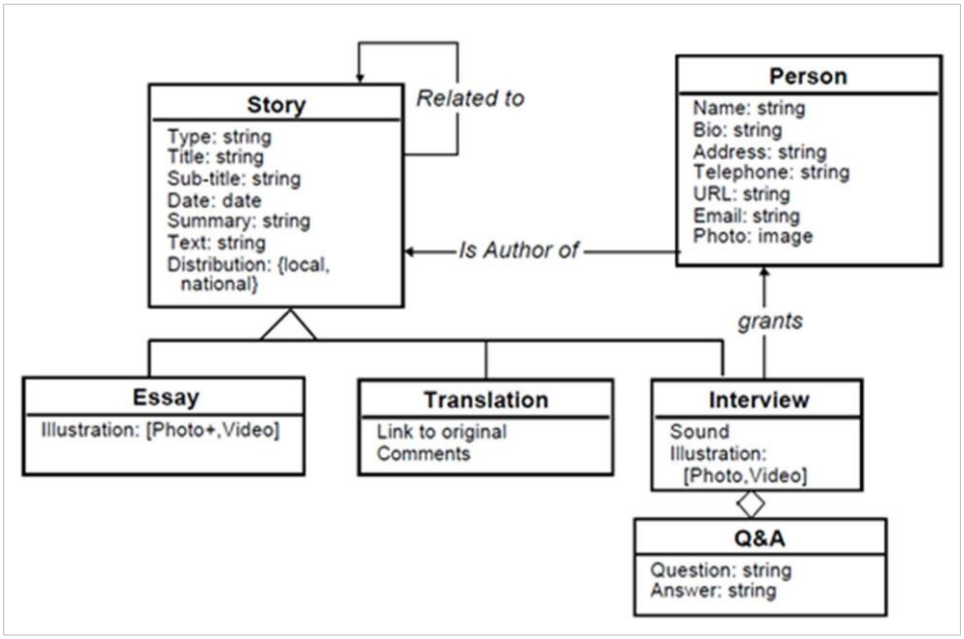


Figura 13. Modelo Conceptual. Revista Online.

✓ **Modelo Navegacional.**

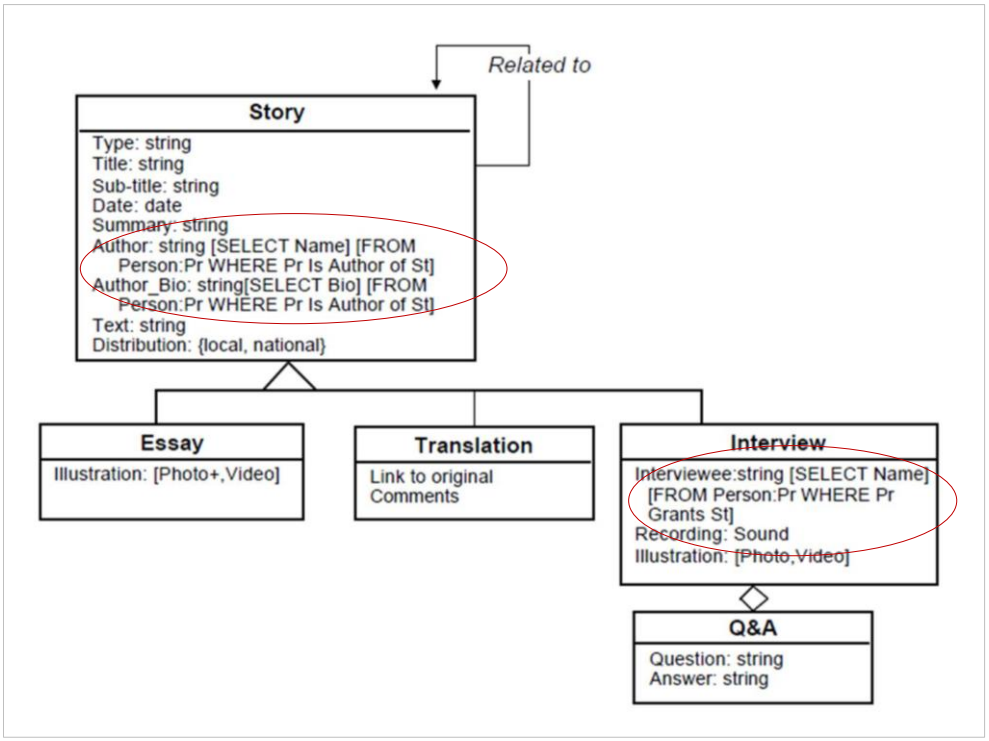


Figura 14. Esquema de Navegación de la Revista Online.

Deberá observarse que en éste esquema no aparece la clase Persona, los atributos authores son ahora parte de la historia. Lo mismo sucede con la persona que concede una entrevista .

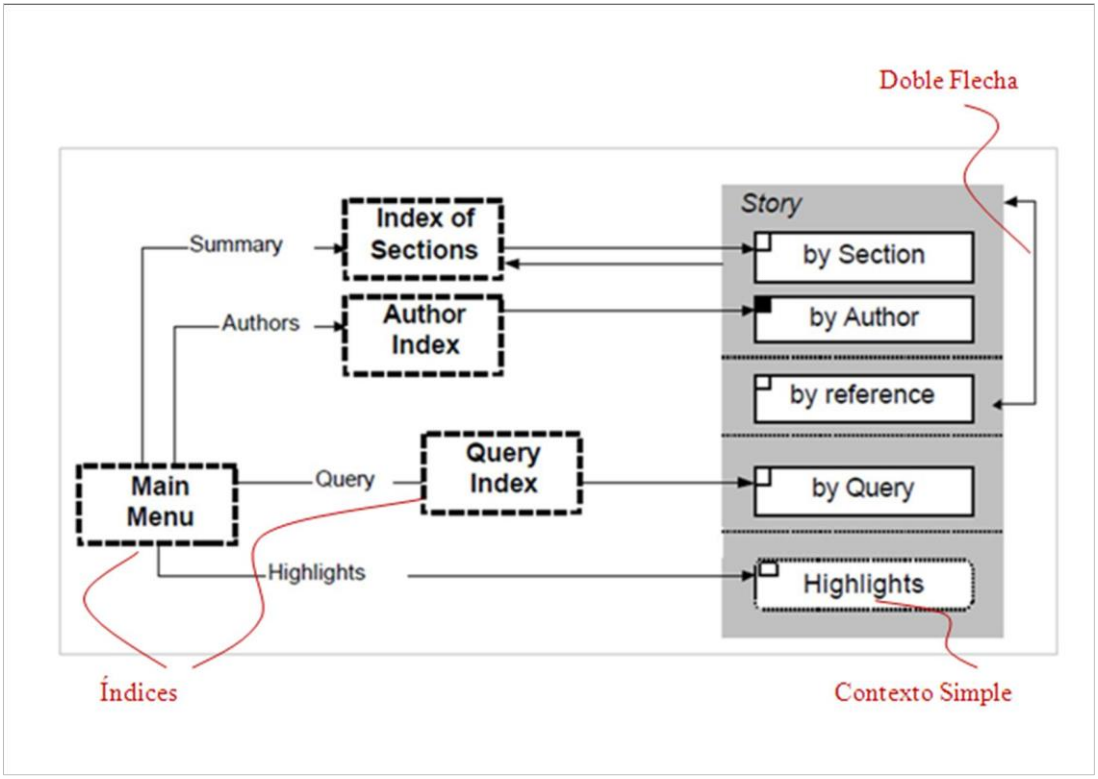


Figura 15. Esquema de Contexto Navegacional.

Los indices se representan con una caja con líneas punteadas gruesas, como se observa en la figura (Main Menu, etc).

El tipo más sencillo de contexto (Conjunto de nodos), se representa como muestra la figura 16.

Figura 16. Notación para un Contexto Simple.

En el ejemplo, un conjunto de historias que se presentan como “highlights” de un determinado tema.

Para representar grupos de contextos que abarquen nodos de la misma clase, de acuerdo con diferentes criterios. (Por ejemplo, las historias pueden organizarse de acuerdo a las diferentes secciones o autores). Se utiliza la siguiente notación:

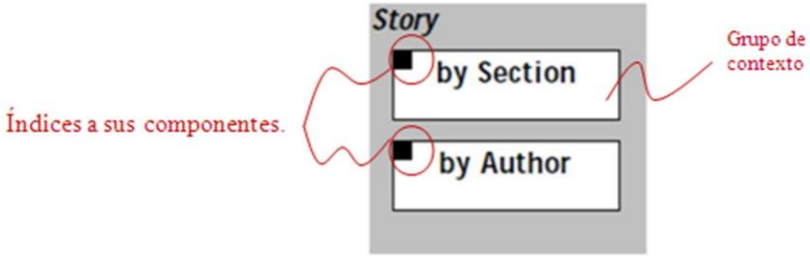


Figura 17. Notación para Grupo de Contextos.

En el ejemplo se puede observar dos grupos de contextos:

- “by Section”: Historias recopiladas por cada sección.
- “by Author”: Obtiene las historias de cada autor.

El cuadrado negro de la esquina superior izquierda indica que el grupo posee un índice a sus componenetes.
En la definición del contexto mismo especificará los tipos de navegación permitidos dentro del contexto, los valores típicos son:

- sequential
- sequential circular
- index (donde es posible navegar sólo a partir del índice de un elemento)
- index sequential

La ausencia de una línea discontinua entre los grupos indica que es posible conmutar automáticamente entre ambos grupos de contextos. (by Section , by Author)
No ocurre lo mismo con los grupos de contexto “by reference”, “by query”, y por el contexto simple “highlights”, los cuales están separados por líneas punteadas.
Se puede comprobar en el ejemplo lo siguiente:

- Si el lector está mirando una historia en una sección determinada, lo hará se le permitirá desplazarse a cualquier “Otra historia próxima en esta section (by Section)” o “a la próxima historia del el mismo author (by Author)”.
- Sin embargo, no se le permitiría navegar “al siguiente highlight”.

La flecha que va desde “Story” a “by Reference” indica que las historias pueden referirse a otras historias que están relacionadas.
Si el lector está mirando una historia y, a continuación, sigue un enlace a una historia relacionada, llegará el contexto en el que se agrupan todas las historias relacionadas. Estando en ese punto, se debe volver ya sea a la historia original, como se indica por la dobleflecha, o al índice de resumen, como esta indicado en el esquema por la flecha de “Story” a la “Index of Sections”.

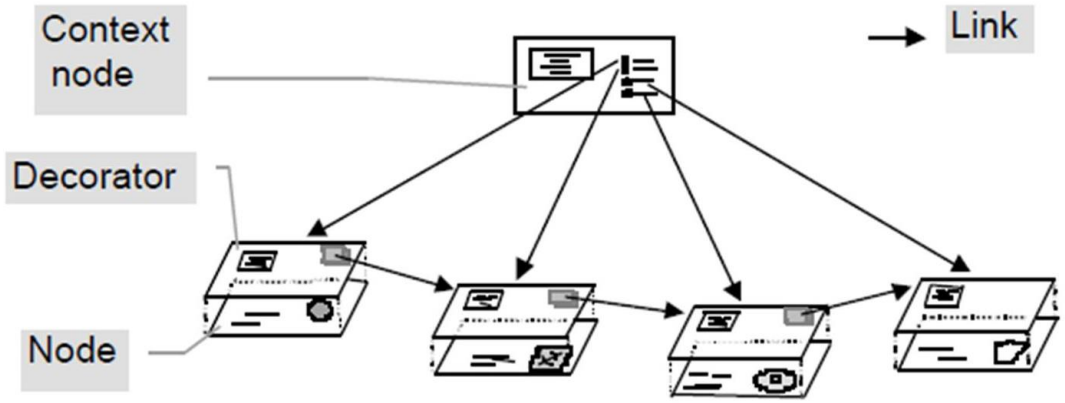


Figura 18. Un Nodo en un Contexto es construido para un nodo básico y un decorator.

Los nodos están enriquecidos con un conjunto de clases especiales (llamadas clases InContext) que permiten que un nodo tenga un aspecto diferente y presentar diferentes atributos (incluyendo enlaces), como también los métodos (comportamiento) cuando se navega dentro de un contexto particular (esto se muestra en la Figura 18).

Siguiendo el ejemplo: cuando se atraviesa “Stories por Bob Woodward”, la vida de los authores no puede ser incluido como un atributo de la historia, mientras que se podría incluir al atravesar “Highlights” ver Figura 19. Este enriquecimiento sigue de cerca la estructura del patrón de diseño decorador [Gamma95].

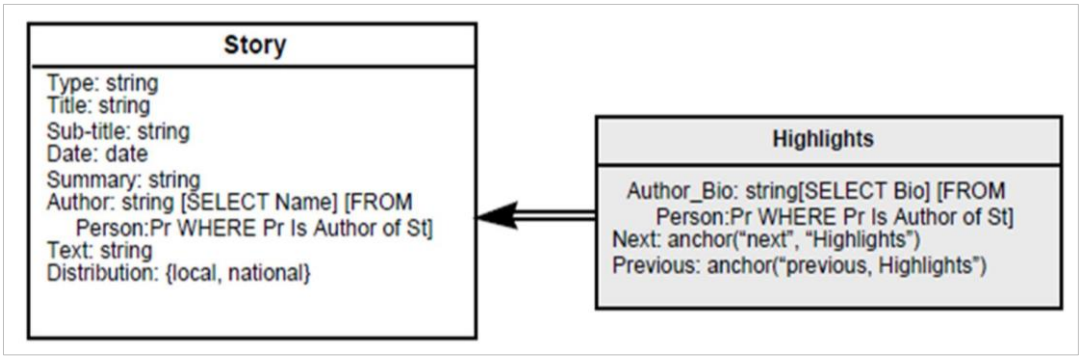


Figura 19. Especificación de la clase InContext de los contextos "Story" y "Highlights".

En este caso, la vida del author sólo se ve cuando se accede a una historia en el contexto "Highlights".

Debe tenerse en cuenta que los enlaces “previous” y “next” también son atributos que pertenecen a la clase InContext “Story” en el contexto “Highlights”. Otros contextos, si tuvieran navegación secuencial, agregarían sus propios enlaces “next” y “previous”. [SCH98].

✓ **Diseño de Interfaces Abstractas**

Una vez definida la estructura de la aplicación se definirán los objetos de la interfaz de navegación: como aparecerán, que funcionalidades activarán y que tranformaciones se realizarán.

Un componenete crítico de las aplicaciones web es el aspecto interactivo de la interfaz de usuario.

En OOHDM, se utiliza el enfoque de diseño “Abstaract Data View” (Vista de datos abastracta), para describir la interfaz de usuario de una aplicación hipermedia [Co 95].

Las ADV como se explicó anteriormente, son objetos que tienen un estado y una interfaz, donde la interfaz puede ser ejecutada a través de mensajes (generalmente, eventos externos generados por el usuario). Sólo representan la interfaz y el Estado, NO la aplicación.

En el enfoque de diseño de las ADV, una ADV puede estar compuesta por la agregación o composición de ADV de un nivel inferior, permitiendo construir interfaces de usuario con objetos anidados.

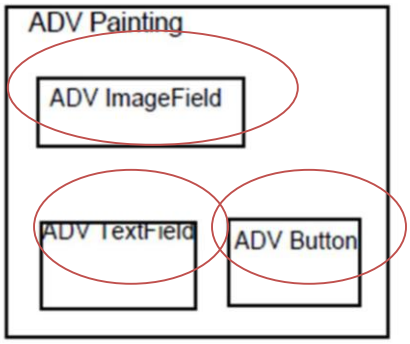


Figura 20. Utilizando agregaciones en las ADV.

En la Figura 20 se muestra cómo para describir una pintura se pueden utilizar una ADV compuesta por tres ADV (una que contiene una imagen, otra que contiene el texto y otra con un botón).

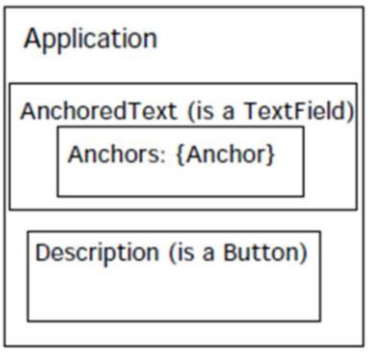


Figura 21. Utilizando Herencia en las ADV.

También se podría organizar en jerarquías de "generalización /especialización", donde se necesite modelar jerarquías de objetos de interfaz. La figura 21 muestra el uso de la "Herencia" en una ADV. "AnchoredText" es un objeto de interfaz que añade un conjunto de enlaces al TextField más general. Mientras que "Description" es un botón especializado que añade un comportamiento personalizado (esto no se muestra en la igura).

Se muestra una ADV que corresponde al diseño del sitio web Portinari (http://www.portinari.org.br/) una aplicación hipermedia que contiene parte de la obra y los documentos relacionados de Candido Portinari, un famoso pintor brasileño.

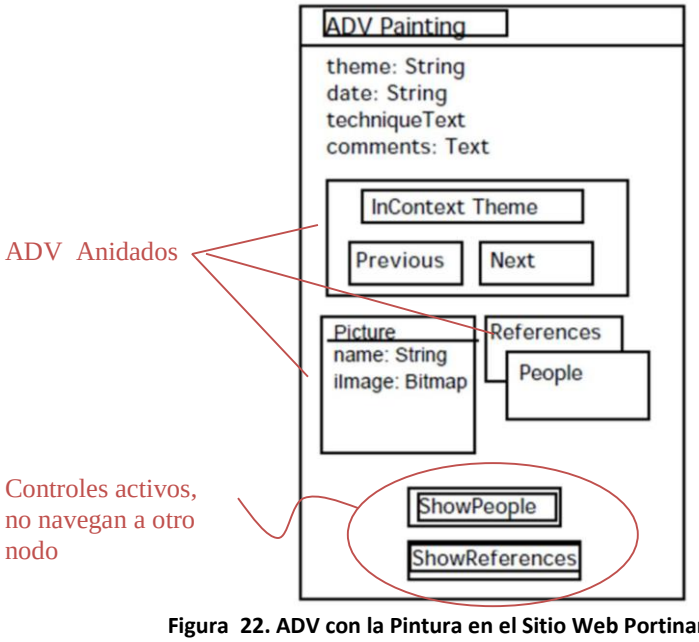


Figura 22. ADV con la Pintura en el Sitio Web Portinari.

"ADV Painting" (Pintura), contiene atributos que describen ciertos aspectos de la pintura y muchos ADV anidados como imágenes, referencias y personas.

En la la figura 22, “showReferences” y “ShowPeople” no están destinadas a ser mostradas en el mismo tiempo y por lo tanto sus ADV se superponen. Estas ADV son controles activos que permiten mostrar una y otra.

ADV “InContext Theme” implementa la interfaz del Tema de Clase InContext y proporciona controles de navegación dentro del contexto de navegación: Pinturas de un tema.

ADV similares deben ser especificados para otros contextos de navegación, tales como pinturas de una Técnica.

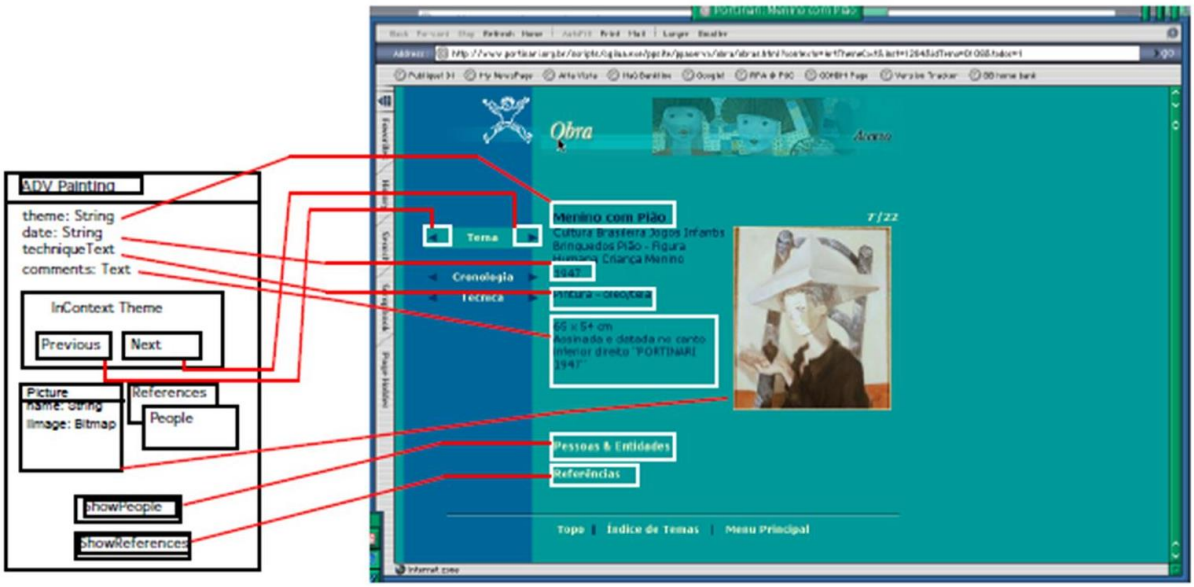


Figura 23. ADV y su relación con los objetos de la interfaz real de la Aplicación.

El diagrama de la figura 22 muestra la naturaleza estática de interfaces Painting’s, mientras que las intefaces dinámicas se muestran en la figura 23 con ADV-gráficos.

En la Figura 23 se muestra en el sitio web Portinari, la interfaz real y la forma en que se relaciona con sus objetos homólogos de interfaz abstracta.

Existe una ligera variación en el gráfico, cuando se hace click en “ShowPeople” envía mensaje de la lista de personas asociadas a la pintura, como ocurre con “ShowRefence”. Esto es particular de éste ejemplo que no implique navegar a otro nodo.

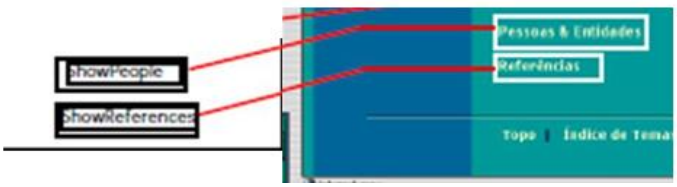


Figura 24. Controles: ShowPeople y ShowRefence.

El objeto de interfaz “Previus” es un objeto de InContext se comunica con el enlace correspondiente y lleva a navegar a otra pintura (la anterior en este ejemplo).

Aun cuando la puesta en marcha de la aplicación no sea orientada a objetos, la implementación se realiza fácilmente en la mayoría de las plataformas.

✓ **Implementación**

- Implementación de Contextos: En este caso, cualquiera de las mejores técnicas conocidas para el mantenimiento del estado de la WWW se puede emplear: estado del paso de la información desde una página a otra (dentro de la URLs), mantener el estado de los campos que pasan escondidos, el uso de las cookies.
- Implementación de la Interfaz: La organización interfaz real y el comportamiento se especifican en las ADV y la distribución física y la apariencia se deben definir en esta fase. Interfaces de cliente en la WWW se pueden implementar usando varias alternativas - simple HTML, HTML con Javascript, HTML con plugins, HTML con Java, Java puro, etc …

La Figura 25 contiene una representación esquemática de la arquitectura del entorno OOHDM-Web. Se pueden utilizar otras plantillas.

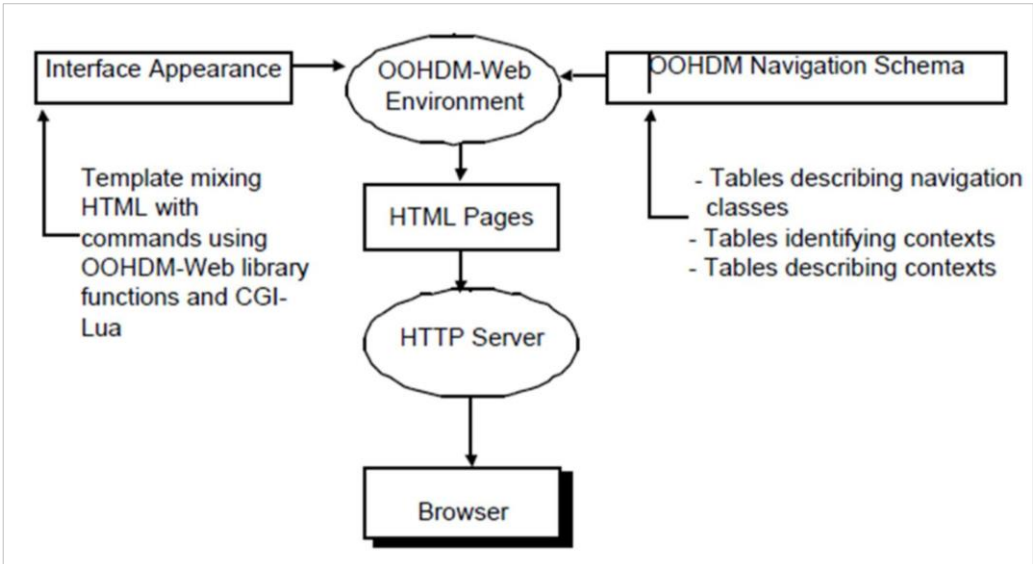


Figura 25. Estructura en un entorno Web OOHDM.

A modo de ejemplo se muestra un diagrama en un entorno Web OOHDM, basado en scripting Lua [LER96] y en el entorno CGILua [HES97] llamó OOHDM-Web [Pontes97].

Este entorno implementa las plantillas que son una mezcla de HTML simple y llamadas a las funciones de una biblioteca que da acceso a los objetos de navegación. Estos objetos se almacenan en una base de datos relacional, que se accede a través de ODBC.

- **Herramienta de Modelado.**
  No se eencontraron herramientas disponibles de modelado con OOHDM.

3.3.1.1.2.3. WebML (Web Modeling Language) ó Lenguaje de Modelado Web

Para modelar las características propias de las aplicaciones Web, las metodologías de diseño y desarrollo de aplicaciones basadas en UML (Booch y otros en el año 1998) no proveen el soporte específico, ya que las aplicaciones Web deben cumplir con muchos tipos de requerimientos (como ser escalabilidad, seguridad, portabilidad del software del cliente, entre otros) y además deben proveer una generación dinámica de la interfaz de usuario, características avanzadas de presentación y ser fáciles de personalizar.

En las últimas décadas han sido desarrolladas muchas metodologías y herramientas para modelar y diseñar aplicaciones web, ver tabla 4.

Tabla 4. Comparación de Metodologías y sus características.

METODOLOGÍA O HERRAMIENTA DE MODELADO	AUTORES	PRINCIPALES CARACTERÍSTICAS
RMM (Relationship Management Methodology o Metodología de Administración de Relaciones)	Creada por Isakowitz, Stohr y Balasubramanian en 1995.	Se define como un proceso de análisis, diseño y desarrollo de aplicaciones hipermedia. Los elementos principales de este método son el modelo E-R (Entidad-Relación) y el modelo RMDM (Relationship Management Data Model) basado en el modelo HDM. Esta metodología es apropiada para dominios con estructuras regulares (es

METODOLOGÍA O HERRAMIENTA DE MODELADO	AUTORES	PRINCIPALES CARACTERÍSTICAS
		decir, con clases de objetos bien definidas, y con claras relaciones entre esas clases). Por ejemplo, catálogos o "frentes" de bases de datos tradicionales. Según sus autores, está orientada a problemas con datos dinámicos que cambian con mucha frecuencia, más que a entornos estáticos.
OOHDM (Object Oriented Hypermedia Design Method o Método de Diseño de Hipermedia Orientado a Objetos) [CAS07]	Fue diseñado por D. Schwabe, G. Rossi, y S. D. J. Barbosa en 1996.	Es una extensión de HDM con orientación a objetos, que se está convirtiendo en una de las metodologías más utilizadas. Ha sido usada para diseñar diferentes tipos de aplicaciones hipermedia como galerías interactivas, presentaciones multimedia y, sobre todo, numerosos sitios web. Al igual que RMM, este método se inspira en el modelo HDM, pero lo que le distingue claramente del primero es el proceso de concepción orientado a objetos. OOHDM propone el desarrollo de aplicaciones hipermedia mediante un proceso de 4 etapas: diseño conceptual, diseño navegacional, diseño de interfaces abstractas e implementación. Cada etapa de la concepción define un esquema objeto específico en el que se introducen nuevos elementos (clases).
UWE (UML-based Web Engineering) [CAS07]	Desarrollado por Conallen en 1999 y por Koch y Hennicker en 2000.	UWE es un método de ingeniería del software para el desarrollo de aplicaciones web basado en UML. Cualquier tipo de diagrama UML puede ser usado, porque UWE es una extensión de UML.

METODOLOGÍA O HERRAMIENTA DE MODELADO	AUTORES	PRINCIPALES CARACTERÍSTICAS
		UWE utiliza "pura" notación UML y tipos de diagramas UML siempre que sea posible para el análisis y diseño de aplicaciones Web, es decir, sin las extensiones de cualquier tipo. Para las funciones Web específicas, como nodos y enlaces de la estructura del hipertexto, el perfil UWE incluye estereotipos, valores etiquetados y restricciones definidas para los elementos de modelado. La extensión UWE cubre navegación, presentación, procesos de negocio y aspectos de adaptación. La notación UWE se define como una "ligera" extensión del UML.
Autoweb	La mayor parte de su desarrollo la ha realizado Fraternali y Paolini en el Instituto Politécnico de Milán en el año 2000.	Propone una metodología y también un entorno de diseño para páginas web con gran cantidad de datos. Las principales características del sistema son: • Adaptarse a los modelos actuales de hipermedia para desarrollo web y a las necesidades de la generación automática de software. • Utilizar la tecnología de las bases de datos no sólo para almacenar el contenido de una aplicación web si no para almacenar una descripción de su estructura, navegabilidad y presentación, para que se permita la implementación automática y una evolución mejor.

METODOLOGÍA O HERRAMIENTA DE MODELADO	AUTORES	PRINCIPALES CARACTERÍSTICAS
		• Definir un proceso de desarrollo software para construir nuevas aplicaciones web y para poder practicar ingeniería inversa sobre aplicaciones basadas en base de datos ya existentes. • Dar soporte a estos procesos mediante herramientas sencillas.
WebML (Web Modeling Language o Lenguaje de Modelado Web)	Creado por Ceri y otros en el año 2000.	Es una notación visual para el diseño de aplicaciones Web complejas que usan datos intensivamente. Provee especificaciones gráficas formales para un proceso de diseño completo que puede ser asistido por herramientas de diseño visuales.
OO-H (Hipermedia Orientada a Objetos)	Desarrollado por Gómez, Jaime y por Cachero, Cristina en la Universidad de Alicante, España, en 2003.	Proporciona un conjunto de nuevas vistas que extienden al UML para proporcionar un modelo de interfaz Web. El proceso de generación de código puede, partiendo de tales diagramas y sus valores asociados etiquetados, generar una interfaz web capaz de conectar a los módulos de negocio subyacentes.
WSDM (Web Services Distributed Management)	Fue aprobado como estándar OASIS en marzo de 2005.	Es un Método de Diseño para Sitios Web, donde hay un acercamiento al usuario que define los objetos de información basado en sus requisitos de información para el uso de la Web. En este método se definen una aplicación Web a partir de los

METODOLOGÍA O HERRAMIENTA DE MODELADO	AUTORES	PRINCIPALES CARACTERÍSTICAS
		diferentes grupos de usuarios que vaya a reconocer el sistema. Propone cuatro etapas: modelo de usuario, diseño conceptual, diseño de la implementación e implementación. El tratamiento de requisitos se lleva a cabo en la etapa inicial, donde, en primer lugar, se identifican y clasifican los usuarios que van a hacer uso de la aplicación Web.
WA-UML (Web Adaptive Unified Modelling Language) [RAO06]	Desarrollado por Raoudha Ben Djemaa, Ikram Amous, Abdelmajid Ben Hamadou, en el Laboratoire MIRACL en Francia para el Eighth IEEE International Symposium on Web Site Evolution (WSE'06) en Septiembre de 2006.	Es un perfil UML para Aplicaciones Web Adaptables (AWA). Aumenta la expresividad de UML al tiempo que añade etiquetas y anotaciones gráficas para diagramas UML. Esta extensión de UML define un conjunto de estereotipos y limitaciones, que hacen posible el modelado de AWA. Estos estereotipos y las limitaciones de se aplican en un número de diagramas representados en el mismo modelo y en los mismos diagramas que describen el sistema.

- **WebML: Web Modeling Language**

  WebML [CER00] es un lenguaje modelado de alto nivel para la especificación de aplicaciones Web basado en cuatro perspectivas: modelo estructural, modelo del hipertexto, modelo de presentación y modelo de personalización.

En la tabla 5 se muestra brevemente las características principales de cada una de estas perspectivas.

Tabla 5. Características de WebML

<table>
<tr><th>PERSPECTIVA</th><th colspan="2">PRINCIPALES CARACTERÍSTICAS</th></tr>
<tr><td>MODELO ESTRUCTURAL</td><td colspan="2">Expresa el contenido de datos del sitio, en términos de las entidades y relaciones pertinentes. WebML no propone un nuevo lenguaje para el modelado de datos, pero es compatible con notaciones clásicas como el modelo E / R, el modelo ODMG orientado a objetos, y diagramas de clases UML.
Para hacer frente a la exigencia de expresar información redundante y calculada, el modelo estructural también ofrece una forma simplificada, con el OQL-like, un lenguaje de consulta, por lo que es posible especificar la información derivada.</td></tr>
<tr><td rowspan="2">MODELO DEL HIPERTEXTO</td><td colspan="2">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.</td></tr>
<tr><td>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,</td><td>MODELO DE NAVEGACIÓN

Expresa cómo el número de páginas y unidades de contenido que se unen para formar el hipertexto. Los enlaces son no contextuales, cuando conectan semánticamente páginas independientes, o</td></tr>
</table>

PERSPECTIVA	PRINCIPALES CARACTERÍSTICAS	
	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. Las unidades de composición se definen en la parte superior del esquema de estructura del sitio; el diseñador dicta la entidad subyacente o de la relación en la que se basa el contenido de cada unidad.	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.
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 abstracto. 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.	
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	

PERSPECTIVA	PRINCIPALES CARACTERÍSTICAS
	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-like 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.

WebML define también un proceso iterativo en el proceso de diseño, con las siguientes etapas: recolección de requisitos, diseño de datos, diseño del hipertexto, diseño de presentación, diseño de usuarios y de grupos y diseño de personalización.

En la tabla 6 se describe brevemente a cada una de las etapas del proceso iterativo.

Tabla 6. Etapas con sus principales características.

ETAPAS	PRINCIPALES CARACTERÍSTICAS
Recolección de requisitos	Se especifican los requisitos de la aplicación tales como, usuarios a los que está dirigida, ejemplos de contenido, guías de estilo, personalización requerida, restricciones debido a datos heredados, etc.
Diseño de datos	Se realiza mediante un modelado estructural identificando entidades y relaciones. WebML permite la utilización en esta etapa de modelos E-R (Entidad-Relación) o modelos de clases UML.
Diseño del hipertexto	Se realiza el modelado de la estructura del hipertexto, mediante la construcción de dos modelos: el modelo de composición, en el que se identifican qué páginas componen el hipertexto y con qué contenido; y el modelo de navegación que expresa cómo se enlazan las páginas para formar el hipertexto. Para estos modelos, WebML define una notación propia.
Diseño de presentación	Implica la realización del modelo de presentación en el que se representa la apariencia gráfica de las páginas.
Diseño de usuarios y de grupos	Se realiza un modelado de usuarios, donde se identifican los diferentes perfiles de usuarios y grupos y sus necesidades de personalización.
Diseño de personalización	Permite la definición de perfiles para el acceso de los grupos de usuarios identificados.

En la figura 26 [CER00] se observa un esquema de la estructura simple de la publicación de los álbumes y de la información de los artistas. Artista publica álbumes compuestos de canciones, y tienen información biográfica y reseñas de su obra. Para publicar esta información como un hipertexto en la Web, es necesario especificar los criterios para la composición y la navegación, es decir, para definir una vista del sitio.

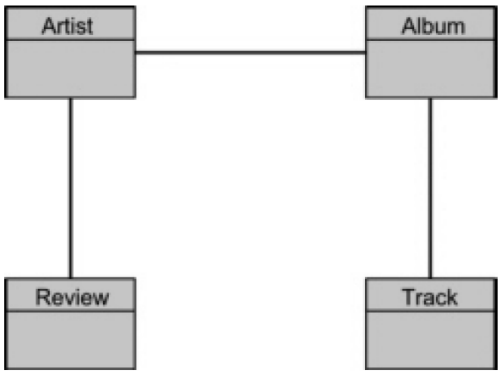


Figura 26. Ejemplo de Esquema de Estructura.

La figura 27 [CER00] muestra un extracto de una especificación de una vista del sitio, utilizando el lenguaje gráfico WebML. El hipertexto se compone de tres páginas, que se muestran como rectángulos de puntos. Cada página encierra un conjunto de unidades (rectángulos sólidos con diferentes iconos) que se muestran juntos en el sitio. Por ejemplo, la página AlbumPage recoge la información de un álbum y de su artista. Contiene una unidad de datos (AlbumInfo) que muestra la información sobre el álbum, una unidad de índice (TrackIndex) que muestra la lista de las canciones del álbum y otra unidad de datos (ArtistInfo) que contiene la información esencial sobre el artista del álbum. La unidad AlbumInfo se conecta con la unidad ArtistInfo por una unidad intermedia directa (ToArtist), lo que significa que AlbumInfo se refiere al (único) artista que compone el álbum que se muestra en la página. La unidad ArtistInfo tiene un enlace de salida que conduce a una página independiente que contiene la lista de revisión, y un enlace a una unidad que apunta directamente a los datos biográficos del artista, mostrados en una página aparte. La figura 27 muestra una posible interpretación de la página HTML AlbumPage.

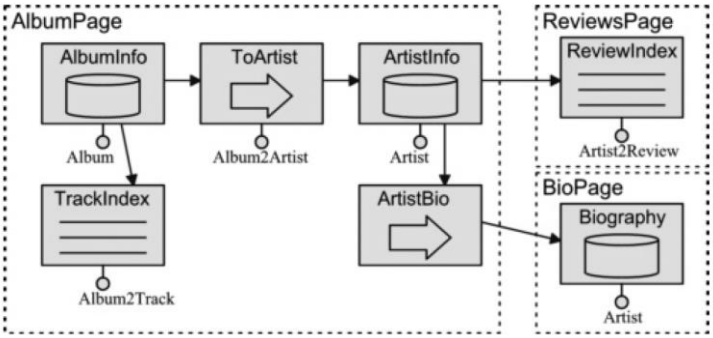


Figura 27. Ejemplo de especificación para composición y navegación

La figura 28 [WRI08] muestra el modelado del hipertexto de una aplicación Web para una tienda de venta de CD´s online.

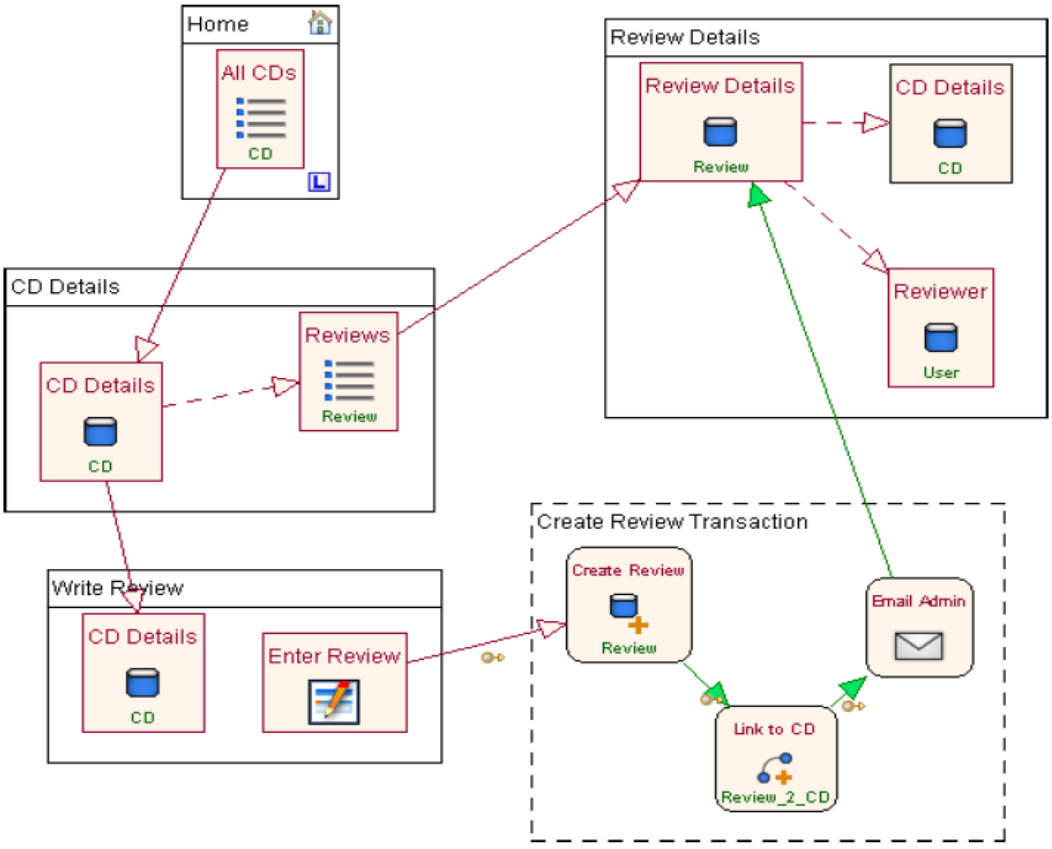


Figura 28. Simple modelo de Hipertexto.

✓ **Características de WebML**

La tabla 7 muestra una comparación realizada por [WRI08] de las características de WebML con algunas de las herramientas o metodologías de modelado de aplicaciones Web nombradas anteriormente.

Tabla 7. Características de WEbML.

Características	WebML	UWE	OOHDM
Eventos	Ok	-	-
Control del Navegador	Pobre	-	-
Ciclos de vida	Pobre	Bueno	-
Usuarios	Bueno	Pobre	-
Seguridad	Ok	Ok	-
Bases de datos	Bueno	Ok	Pobre
Mensajería	Bueno	Pobre	-
Modelado de Interfaz de la Usuario	Pobre	Ok	Ok
Plataforma independiente	Excelente	Excelente	Bueno
Estándares	Pobre	Excelente	Pobre
Meta-modelos	Pobre	Excelente	-

✓ **Eventos**

Ceri [CER02] extiende WebML para manejar operaciones y cadenas de operaciones, lo que permite a WebML describir eventos del lado del servidor. Bozzon [BOZ06] amplía aún más el modelo para permitir distinguir entre las operaciones del cliente y el servidor y los objetos con el símbolo C (Cliente). Los eventos pueden sólo ejecutarse explícitamente de la solicitud, y no pueden real operaciones adicionales en paralelo, excepto a través de servicios web externos [MAN05]. También [BRA05] propone el manejo de las excepciones a través de una variedad de modelos de eventos, sin embargo esto aún no es soportado por el entorno de desarrollo ágil WebRatio.

La figura 29 [WRI08] muestra un ejemplo con extensiones de WebML para Cliente/Servidor.

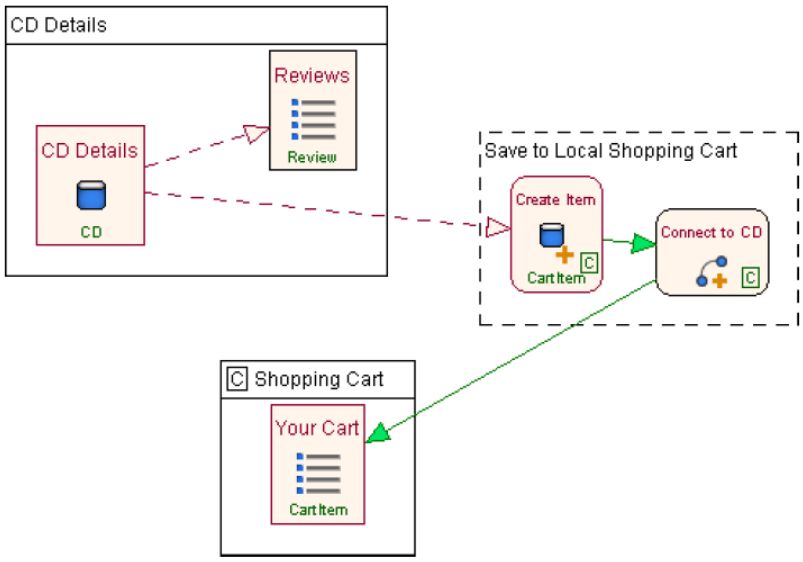


Figura 29. Extensiones WebMl Cliente/servidor

✓ **Control Del Navegador**

Un modelo WebML tiene poco soporte para el control del navegador, ya que no tiene soporte para cookies, scripts o identificación del usuario. Por otra parte permite a los componentes scripts, pero el modelo no puede describir a los scripts en sí mismos.

✓ **Ciclos De Vida**

Los ciclos de vida se pueden considerar para el uso de cadenas de operaciones y enlaces de navegación, pero estos sufren los mismos problemas que el modelado de un evento en WebML. Un ejemplo de este soporte para los ciclos de vida podría ser para eliminar el carrito de la compra de los usuarios al final de sus sesiones. Cabe aclarar que esta cadena de operaciones sólo se produce cuando el usuario cierra la sesión explícitamente. WebML no trata a los ciclos de vida como objetos conceptuales diferentes, ya que de esta manera podría añadir un montón de complejidad e información redundante al modelo.

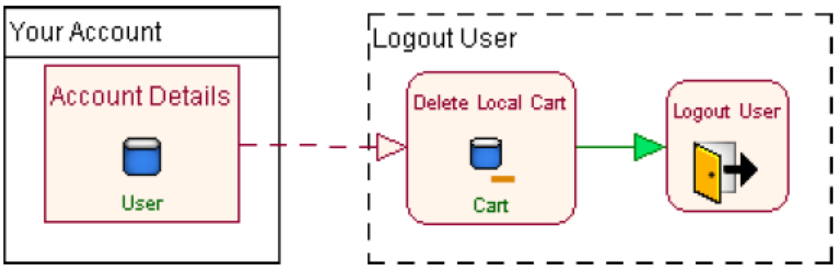


Figura 30. Simulación del Ciclo de vida de un evento en WebMl

✓ **Usuarios y Seguridad**

WebML ha incorporado soporte para usuarios y grupos, pero sólo como miembros del modelo estructural. Cada usuario pertenece a al menos un grupo, y cada grupo sólo pueden acceder a una vista del sitio. El modelo supone que sólo un usuario interactúa con el modelo en un momento dado, por lo que no soporta el modelado de la interacción entre múltiples usuarios. El único permiso representado en el modelo es la capacidad de los grupos para ver determinadas páginas; permisos adicionales de seguridad deben comprobarse explícitamente a través de cadenas de operaciones.

✓ **Bases de Datos**

El uso de un modelo de Entidad-Relación en el modelo estructural de un modelo WebML abstrae los datos del sistema de la base de datos; los modelos de hipertexto pueden acceder en forma abstracta a las entidades en el modelo. WebML cuenta con soporte limitado para subir archivos o manejar múltiples bases de datos, pero debería ser extensible.

✓ **Mensajería**

El soporte para mensajería se limita a la información contextual pasada entre unidades del modelo de hipertexto; las unidades también pueden acceder a los parámetros globales. Ceri [CER07] agrega unidades para permitir al modelo acceder a los parámetros de consulta directamente. WebML tiene un excelente soporte para describir una amplia gama de servicios web [MAN05], y también tiene un buen soporte para el envío de correos electrónicos con muchas partes con archivos adjuntos utilizando su unidad Power Mail.

- ✓ **Modelado de la Interfaz De Usuario**
  No existe un modelo formal para el modelado WebML de la interfaz de usuario, el modelo de presentación genera páginas XML y las transforma mediante hojas de estilo XSL en páginas HTML.

- ✓ **Estándares**
  WebML es independiente de la plataforma de diseño. Las páginas generadas pueden estar en cualquier lenguaje, limitado sólo por la aplicación de la herramienta de software. El modelo WebML es generalmente propietario, excepto para el uso de los diagramas ER para su modelo estructural. No tiene soporte para herramientas de meta-modelado o estándares, y el modelo básico WebML sólo recientemente ha sido implementado como un perfil de UML 2.0 por Moreno en el año 2006. Además carece de soporte integral para los conceptos MDA.
  Esta metodología está siendo actualmente reemplazada por una ifML.

3.3.1.1.2.4. ifML

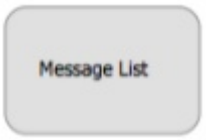
IFML (Interaction Flow Modeling Languaje) nace en el 2013 como estándar surgido del OMG, está basado en WEBML. El objetivo de este lenguaje es representar [SIL13]: (a) La visualización de los contenidos de las interfaces de usuarios; (b)Patrones de navegación; (c) Eventos de usuario y su interacción; (d) Binding a la lógica de negocio (d) Binding a las capas de persistencia; del front-end de las aplicaciones pertinentes a distintos dominios funcionales.

Las ventajas del lenguaje son [SIL13]:

- Especificación formal de las diferentes perspectivas del front-end
- Aislamiento de las problemáticas de la interfaz de usuario
- Separación de los conceptos (interacción de usuario vs backend)
- Simplificación de la comunicación entre expertos de interface de usuario y los stakehokders no técnicos
- Generación automática del código para el front-end de las aplicaciones

La tabla 8 muestra las construcciones gráficas que utiliza este lenguaje.

Tabla 8. Construcciones gráficas utilizadas por IFML

Elemento	Construcción Gráfica
Contenedor	
Componente de Vista	
Evento	
Acción	
Flujo de Navegación	
Flujo de Dato	
Vinculación de Parámetros	

- ✓ **Modelado con IFML**

La figura 31 (tomada de [IFM13a]) permite observar como es la apariencia de un diagrama IFML.

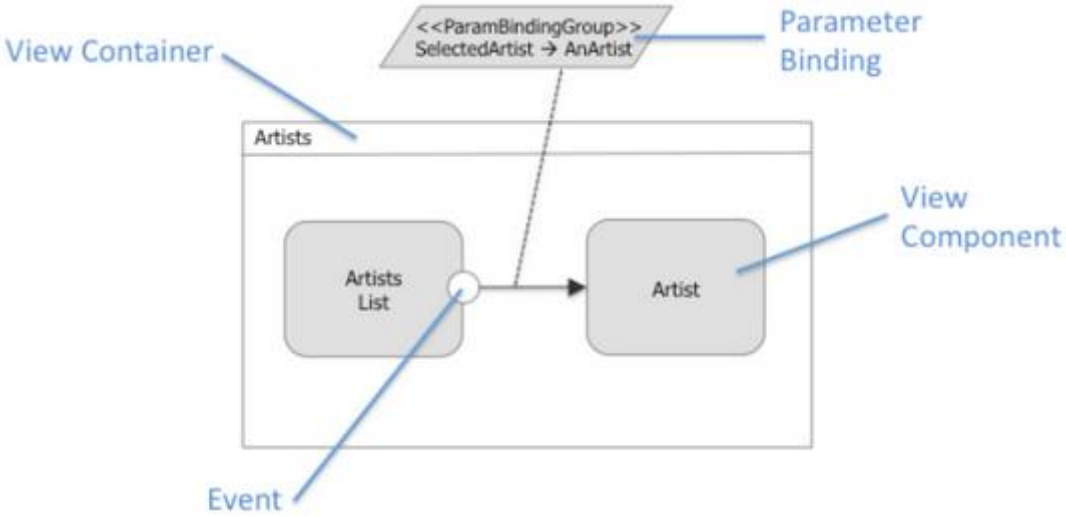


Figura 31. Aspecto de un diagrama IFML- Elementos básicos.

Para entender la sintaxis de este lenguaje resulta interesante el diagrama presentado en la figura 32 [SIL13].

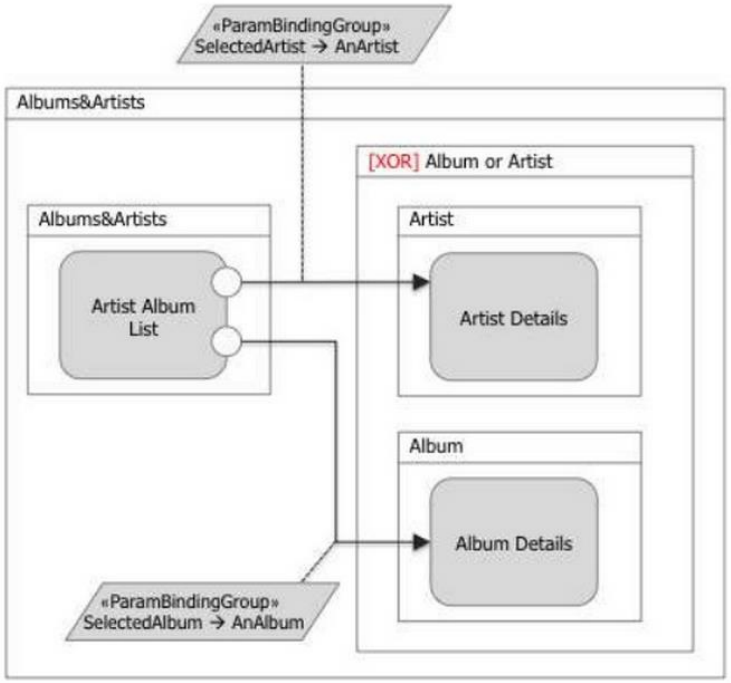


Figura 32. Aspecto de un diagrama IFML- sintaxis

Algo que resulta de interés es que pueden agregarse detalles a los ViewComponentes como se muestra en la figura 33 tomada de [SIL13].

Figura 33. Agregar detalles a un ViewComponentes.

En [SIL13] se indica que también es posible contar con distintos subtipos de componentes tal como lo muestra la figura 33. También dentro de la representación gráfica para un evento (un circulo) es posible agregar algún dibujo particular para identificar de que se trata el evento a modo de ejemplos ver la figura 34, según el dominio en cuestión podrá variarse el objeto a dibujar para hacer más especifico el evento.

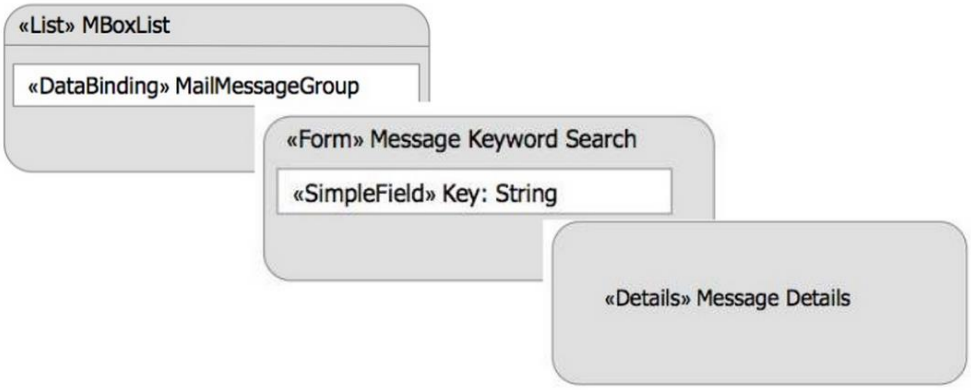


Figura 34. Ejemplos de Subtipos de Componentes.

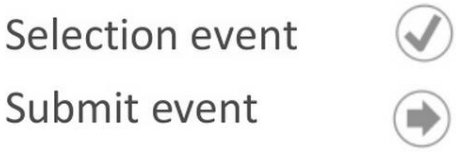


Figura 35. Ejemplos de Subtipos de Eventos

En la figura 36 (tomada de [IFMc]) se presenta un diagrama más complejo que el presentado anteriormente en el que pueden observarse todos los elementos gráficos disponibles en el lenguaje presentados previamente en la tabla 8.

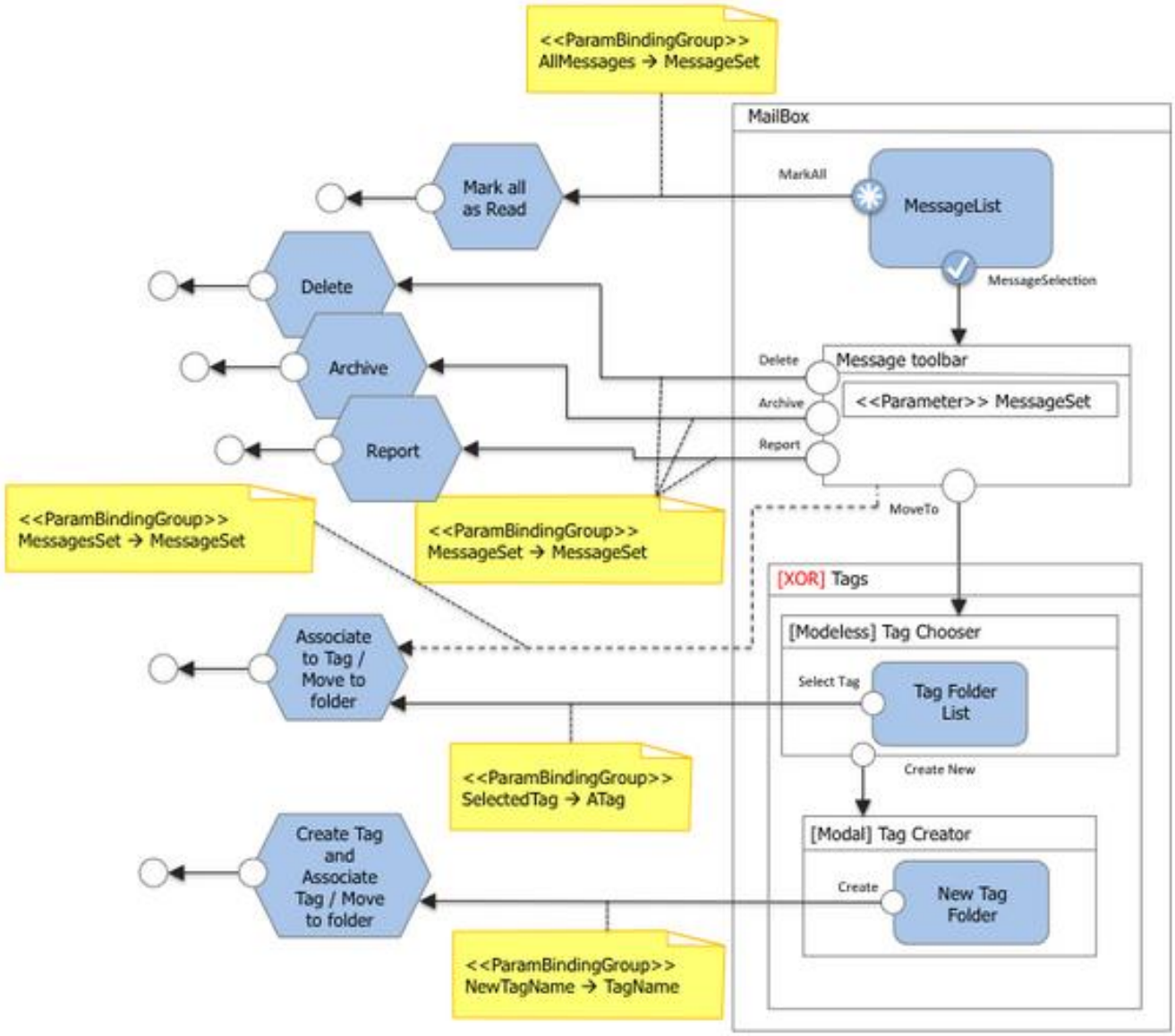


Figura 36. Diagrama IFML avanzado.

Se cuenta en internet con algunos ejemplos modelados completos con IFML como ser el Modelado de una empresa de ventas de libros por internet [IFM13b].

3.3.1.1.2.5. UWe

UWe (UML Based Web Ingeneering) es una extensión del meta model UML 2.0. Por ésta razón contiene todos los elementos de UML. Y los elementos que UWe agrega están relacionados con los de UML. Es una metodología basada en el Proceso Unificado. El objetivo principal de su creación fue encontrar una metodología estandar para analizar y diseñar modelos de sistemas web. Por eso se adhirió a UML. [ROS08]

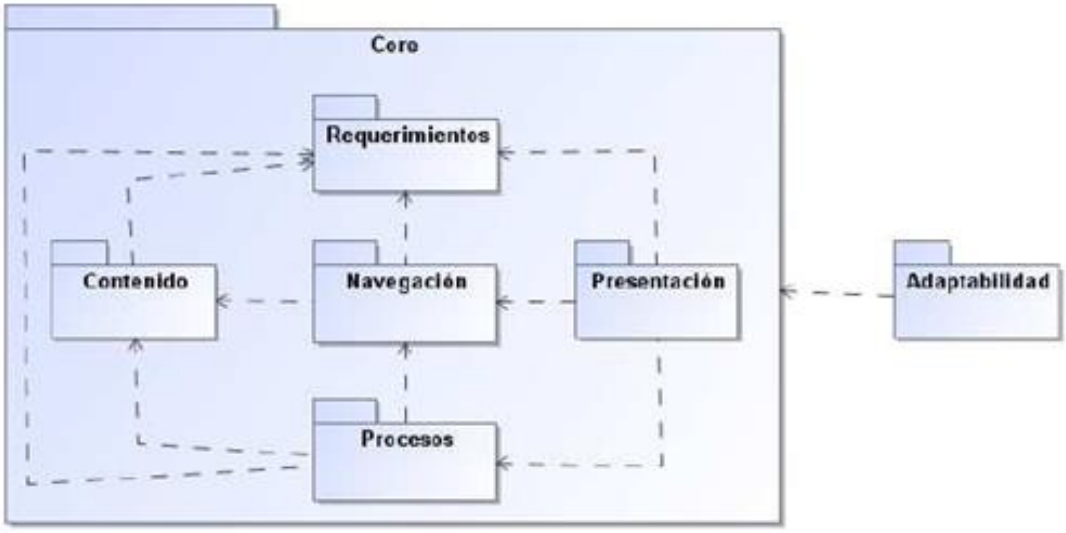


Figura 37. Meta Modelo [ROS08]

"Las fases que utiliza la metodología UWE son: Análisis, Diseño e Implementación. La metodología no define una fase clara para realizar el levantamiento de requerimientos, por lo tanto en el desarrollo de este proyecto se decidió utilizar un estándar como la IEEE 830 para definir los requerimientos funcionales y no funcionales. Gracias a este estándar se puede tener una fase de análisis más completa porque esto se une al modelo de casos de uso." [NAR12]

"En el ámbito del desarrollo web no es usual modelar mucho las aplicaciones. Quizá es una de las razones por las que los desarrollos se tornan más complejos de lo pensado. La mayoría de los proyectos complejos ya sean estos basados en web o de otro tipo, el cliente espera ver resultados rápidamente, de modo que se suele desestimar la importancia del buen análisis y modelado. Esta es una muy mala práctica, tomando en cuenta que muchas de las aplicaciones que se desarrollan hoy día y que interactúan en la red son sistemas de complejidad media o alta con la salvedad que opera sobre una plataforma web." [ALV09]

3.3.1.1.3. Investigaciones académicas de otros grupos. Trabajos Relacionados.

- **Perfil UML para el desarrollo de aplicaciones WAP [SOT05].**

  - **Vinculación:**
    Este artículo se relaciona con el concepto de UML, ya que utiliza una extensión del mismo (la cual se conoce como "Perfil"), con el fin de darle las herramientas necesarias para poder realizar aplicaciones WAP.

  - **Resumen:**
    El trabajo presenta el concepto de Perfil UML, el cual permite "extender" UML para otorgarle a los desarrolladores las herramientas necesarias (clases, estereotipos, valores etiquetados y restricciones) para modelar aplicaciones WAP (Wireless aplication protocol). Se presentan conceptos tales como el de METAMODELADO, el cual es un mecanismo que permite construir formalmente lenguajes de modelado, y el cual consiste en la definición de todos sus elementos, mediante conceptos y reglas de cierto metalenguaje. Para estandarizar todos los conceptos de modelación, desde los modelos más abstractos a los metamodelos, OMG (Object Management Group) propone una arquitectura de 4 capas, la cual consta con 4 niveles llamados M0 (instancias), M1 (modelo del sistema), M2 (el metamodelo) y M3 (el meta-metamodelo). Respecto el desarrollo de una aplicación WAP, es necesario contar con un lenguaje de modelado, tal como UML. Este se usará principalmente en el desarrollo de etapas importantes del proceso, pero no permite llevar a cabo aspectos de diseño, navegación y construcción. Es por eso que se extiende UML mediante el uso de un perfil, con el fin de la construcción de aplicaciones WAP. Este perfil se construye mediante un proceso de 5 etapas, y está compuesto por elementos tales como Estereotipos (representan las nuevas características que son agregadas al metamodelo UML para extender este lenguaje), Valores etiquetados (Son meta-atributos que se asocian a una metaclase del metamodelo extendido por un perfil) y Restricciones (Las restricciones se asocian a los estereotipos e imponen condiciones sobre los elementos del metamodelo que han sido estereotipados). Las 5 etapas mencionadas para realizar la construcción de un perfil son:

- Creación del Metamodelo: en esta etapa se debe definir el metamodelo de la plataforma o dominio de aplicación que se pretenda estudiar, en este caso, aplicaciones WAP.
- Definición del perfil UML: en esta instancia se lleva a cabo la definición del perfil. Para esto se debe crear un estereotipo por cada elemento del metamodelo que deseamos incluir en el perfil.
- Identificar los elementos a extender: en esta etapa se identificar los elementos de UML que se van a extender con cada estereotipo.
- Definición de Valores etiquetados: Por cada atributo presente en el metamodelo se debe agregar el valor etiquetado correspondiente en el perfil UML
- Definición de restricciones: En esta etapa se definen las restricciones que forman parte del perfil.

➢ **Extensión UML para Casos de Uso Reutilizables en entornos Grid Móviles Seguros [ROS09].**

- **Vinculación:**
  En este artículo se hace mención del concepto de UML, el cual será utilizado para poder desarrollar, de una manera sistemática, aplicaciones para casos de uso Grid.

- **Resumen:**
  El artículo presenta una metodología basada en el uso de Perfiles de UML para el desarrollo sistemático de aplicaciones para casos de uso Grid reutilizables, considerando la incorporación de los dispositivos móviles como un recurso más del Grid y aspectos tales como la seguridad y las limitaciones de dichos dispositivos móviles. La extensión de UML define los estereotipos que detallan los casos de usos Grid y los casos de uso de seguridad, los cuales sirven de ayuda en la construcción de diagramas de casos de uso para este tipo de sistemas. Dicha extensión de UML se llama GridUCSec-Profile, la cual está dividida en dos paquetes (GridUCSec y TypesGridUCSec) que contienen los estereotipos para la representación de casos de uso Grid y casos de uso de seguridad. El paquete GridUCSec contiene casos de uso Grid, casos de uso de seguridad, casos de mal uso, asociaciones de permiso, protección, amenaza y mitigación, junto con los actores involucrados, todos necesarios para mejorar los aspectos de seguridad para sistemas Grid. El

paquete TypesGridUCSec define los tipos de datos para los valores de los estereotipos de GridUCSec-Profile, como son el nivel de protección y riesgo, tipos de permiso, de requisito, de activos, de ataque, etc. Se debe tener en cuenta que, para construir un perfil UML concreto y correcto, se deben definir ciertas restricciones y se deben describir, de forma detallada, las relaciones entre casos de uso. Es por eso que, luego de identificar a los estereotipos y sus relaciones entre ellos, se debe realizar un descripción detallada de cada uno de ellos, que describa sus valores etiquetados (atributos asociados al estereotipo; se los debe describir y se debe identificar los valores posibles de cada tipo asociados con él) y que contenga restricciones (limitaciones que tiene el estereotipo y la forma de relacionarse con otros estereotipos y con elementos de UML). Por último, el artículo presenta un caso de aplicación de la extensión GridUCSec-Profile en el ámbito multimedia y de noticias, el cuál busca ofrecer a distintos periodistas, los cuales cuentan con equipos ligeros y están en continuo movimiento, la posibilidad obtener, capturar y transmitir información, manteniendo en todo momento los controles de seguridad necesarios.

- **Ingeniería web dirigida por modelos [SAN10].**

  - **Relación:**
    El artículo se relaciona con el concepto de MDA, ya que utiliza este concepto con el fin de diseñar, desarrollar, mantener, y gestionar diversas aplicaciones relacionadas con la Ingeniería Web, la cual se aplica en ámbitos tales como generación de código para aplicaciones web, modelado conceptual de aplicaciones web, aplicaciones web móviles, accesibilidad para la web, entre otras.

  - **Resumen:**
    El artículo trata sobre el concepto de Ingeniería Web Dirigida por Modelos (MDWE), la cual es una disciplina que aplica el paradigma MDA al diseño, desarrollo y mantenimiento de aplicaciones web de forma independiente de los aspectos tecnológicos de la plataforma, lo cual permite lograr una fácil integración con otros sistemas y, además, poder realizar automáticamente prototipos para evaluar requisitos y realizar pruebas de calidad sobre los modelos definidos. Este procedimiento será aplicado en el ámbito de la ingeniería web, la cual tiene en cuenta aspectos tales como el de aplicaciones

web para la semántica, aplicaciones web móviles, accesibilidad para la web, entre otras. Primero, se presentan dos propuestas de modelado, siendo la primera aquella que se orienta a construir aplicaciones hipermedia en sistemas estáticos. Un ejemplo de esta iniciativa es WebML (Web Modeling Language), la cual es una metodología de modelado visual de aplicaciones web, centrada especialmente en las aplicaciones de uso intensivo de datos. El otro grupo de propuestas es aquel que busca extender los métodos de desarrollo orientados a aplicaciones dinámicas, tratando de introducir la semántica de hipermedia como característica inherente de estas. Este utiliza metodologías tales como UWE (UML-based Web Engineering), la cual sirve para modelar aplicaciones web, y presta una especial atención a la sistematización y personalización (sistemas adaptativos); WSDM (Web Services Distributed Management), la cual es una especificación basada en servicios web para gestionar y monitorizar el estado de otros servicios; y OOWS, la cual es una extensión del método OO-Method, al cual se le añaden a las técnicas de modelado conceptual. Todas estas metodologías necesitan transformar los modelos para generar las aplicaciones web. Según el framework MDA “las transformaciones se pueden aplicar para establecer un proceso de desarrollo trazable desde los modelos abstractos (CIM, PIM) a los modelos dependientes de la plataforma, ó incluso directamente a su implementación.” Las transformaciones se dividen en Transformaciones Modelo-a-Modelo (se divide en Transformaciones verticales, que convierten modelos de un mayor nivel de abstracción en otros de un nivel menor; y Transformaciones horizontales, que describen el mapeo entre modelos del mismo nivel de abstracción) y transformaciones modelo-a-código. Por último, presenta un ejemplo que consiste en el desarrollo de una aplicación web para la compra de discos por internet. El desarrollo de la misma presenta diferentes fases. La primera fase es la de Especificación del Problema, en la cual se generan diagramas de casos de uso a partir de la funcionalidad encontrada en el análisis de los requisitos. Luego sigue la fase de modelado conceptual, en la que se construyen los siguientes modelos: Modelo de Objetos, Modelo Dinámico, Modelo Funcional, Modelo navegacional y Modelo de Presentación. La última fase es la de Desarrollo de la Solución, donde utilizando una estrategia de compilación de modelos, se obtiene el prototipo software completo de manera automática.

- **Intercambio de modelos UML y OO-Method. [MAR07]**

  - **Vinculación:**
    El artículo propone un método que permite transformar los modelos UML en los modelos requeridos por un compilador de modelos OO-Method, para lo cual realiza un proceso de transformación basado en XMI.

  - **Resumen:**
    El objetivo que presenta el artículo es el de la elaboración de un proceso que transforma los modelos UML (construidos con herramientas CASE) de propósitos generales en los modelos conceptuales concretos requeridos por un compilador de modelos OO-Method (un método de producción automática de software a partir de modelos objetuales compatible con MDA) y viceversa. Este artículo presenta la correspondencia entre modelos UML y OO-Method, y una implementación que transforma automáticamente dichos modelos según dicha correspondencia. Luego se presenta la correspondencia entre elementos UML y OO-Method. Para esto, se debe tener en cuenta que OO-Method utiliza elementos pertenecientes a UML. Las correspondencias mencionadas son:
    - Correspondencia de clases:
    - Una clase describe un conjunto de objetos que comparten las mismas especificaciones de características, restricciones y semántica.
    - Correspondencia de atributos:
    - Los atributos de una clase, ya sea UML u OO-Method, representan características de ella, por lo que su correspondencia es directa.
    - Correspondencia de servicios:
    - Los servicios definen el comportamiento de los objetos de una clase.
    - Correspondencia de asociaciones:
    - Las asociaciones representan relaciones entre dos clasificadores, que pueden ser clases, actores, interfaces, etc.
    - Correspondencia de generalización:
    - La generalización corresponde a una asociación entre varios clasificadores generales (hijos) y un clasificador específico (padre). En UML, los hijos heredan todo el comportamiento del padre y agregan información adicional. Sin embargo, en OO-Method la herencia es semánticamente más precisa, pues permite especificar tanto la generalización permanente como la generalización temporal.

- Correspondencia de paquetes:
- Los paquetes son utilizados para agrupar elementos y proveer un espacio de nombres común a los elementos agrupados.

Luego, define dos procesos de transformación, uno de UML a OO-Method, y el otro el proceso de transformación inverso, es decir, de OO-Method a UML.

- ✓ **Proceso de transformación desde UML a OO-Method**

  Para iniciar el proceso de transformación de un modelo UML a uno OO-Method, es necesario volcar el modelo UML en un archivo XMI. A continuación, con el archivo XMI que contiene el modelo UML, se efectúa la transformación automática del modelo UML a un modelo OO-Method, aplicando las correspondencias detalladas. Al término del proceso de transformación, se obtiene como resultado un archivo XML que contiene la definición del modelo equivalente para OO-Method. Finalmente, el archivo XML generado como producto de la transformación del archivo XMI, es importado en la herramienta OO-Method.

- ✓ **Proceso de transformación desde OO-Method a UML:**

  Al igual que en la transformación desde UML a OO-Method, el primer paso en el proceso de transformación es volcar el modelo OO-Method en un archivo XML. Una vez que se tiene el archivo XML que contiene el modelo OO-Method, se realiza su transformación automática a un modelo UML, utilizando las correspondencias ya mencionadas. Se obtendrá un archivo XMI como resultado de la transformación automática. Por último, el archivo XMI generado es importado en la herramienta UML.

3.3.2. Etapas 2 – Modelado

Al momento de especificar un sistema para ser programado es necesario definir que pantallas contendrá, que datos se visualizarán y cómo será la secuencia de navegación dentro del sistema. El objetivo buscado por esta metodología es lograr que esa especificación pueda realizarse en modelos UML de forma completa y que esos modelos puedan ser llevados luego a generar el código fuente completo de la aplicación ya que se brindarán todos los detalles necesarios en el modelo para lograr dicho objetivo.

Se sabe que UML es un lenguaje de representación general, el cual para tal fin cuenta con un vocabulario gráfico. En algunos casos cuando se quiere modelar un tipo de aplicación o dominio particular, el vocabulario gráfico de UML resulta ser muy reducido. Por esta razón es necesario extender el lenguaje con nuevos artefactos que permitan modelar las características particulares de un dominio en cuestión, por lo cual es necesario crear un profile. Este trabajo propone a través de la formalización de un profile la creación de artefactos que permita modelar las características de los sistemas de hipermedia. Uno de los trabajos que explica los conceptos principales es el modelo OOHDM (Object Oriented Hypermedia Design Methodology), desarrollado en el punto 3.3.1.1.3.2.

3.3.2.1. Generación de un Profile (Perfil de UML)

Enmarcando a los sistemas hipermedia dentro de un contexto especifico como por ejemplo el Modelado de Sistemas Hipermedia en forma general, o el Modelado de Sistemas Hipermedia Móvil con un grado de especificación más alto podemos desarrollar un profile, para las características particulares de los mismos, es decir el diseño se debe adaptar a las características particulares de su entorno de diseño y ejecución en el caso móvil sobre todo a las características particulares de su entorno de ejecución.

Uno de los tipos de sistemas hipermedia más difundidos son los sitios web donde el usuario puede visualizar: imágenes, texto, componentes multimedia; además de navegar en el sitio mediante hipertexto. Un caso particular de los sitios web, son aquellos diseñados especialmente para dispositivos móviles. A pesar de la gran evolución de los equipos celulares que cuentan con una alta capacidad de procesamiento, estos siguen teniendo las siguientes limitaciones y necesidades:

- Pantalla de tamaño reducido
- Necesidad de controles simples
- Mostrar la información de forma sencilla y directa, sin las complejas distribuciones de pantalla de los sitios web convencionales
- Sistema de navegación práctico e intuitivo
- Ingreso de texto sencillo

Estas características particulares se formalizan a través de un profile UML. OMG (Object Management Group) ha creado profiles como solución para la falta de elementos de representación de dominios específicos. Un profile es un mecanismo para extender un leguaje a fin de expresar conceptos más específicos de ciertos

dominios de aplicación. Según OMG “un profile es un subconjunto del meta-modelado de UML, este subconjunto del meta-modelado determina las reglas para representar a este subconjunto del UML” [FUE04].

Al momento de modelar utilizando la metodología es necesario que las herramientas que se utilicen (diagrama de clases por ejemplo), sean aptas para modelar el sistema hipermedia, es decir deben permitir modelar el diseño conceptual, el diseño navegacional etc.

Al momento de modelar para este dominio específico surgen problemas en la expresividad de UML, por lo que es necesaria una extensión del lenguaje, permitiendo crear nuevos artefactos ya sea para las tareas específicas ó bien con un significado determinado para el dominio de la aplicación.

UML provee un mecanismo de extensibilidad para poder ampliar el vocabulario, estos mecanismos se encuadran dentro de la definición de profiles.

- Estereotipos: Permiten la creación de nuevos tipos de bloques de construcción que derivan de otros existentes pero no son específicos de un problema particular. Estos son definidos por un nombre y un grupo de elementos del meta-modelado. Los estereotipos representan una nueva característica agregada al UML para extender el lenguaje.
- Valores Etiquetados: Los valores etiquetados son propiedades nuevas para elementos existentes, estos son meta-atributos que son asociados a una meta-clase de un meta-modelado extendido del profile. Cada valor etiquetado tiene un tipo y es asociado a un estereotipo.
- Limitaciones o Restricciones: Forman reglas (de consistencia o de negocios) sobre los elementos y sus propiedades. Las limitaciones son asociadas a los estereotipos, imponen condiciones a los elementos del meta-modelado que fueron estereotipados. Las limitaciones son escritas en un lenguaje natural denominado OCL (Object Contraint Language)

Se utiliza una extensión del lenguaje para definir ciertas características necesarias sobre los diagramas de clases y componentes. Esta extensión del lenguaje está definida mediante el perfil UML de la figura 38.

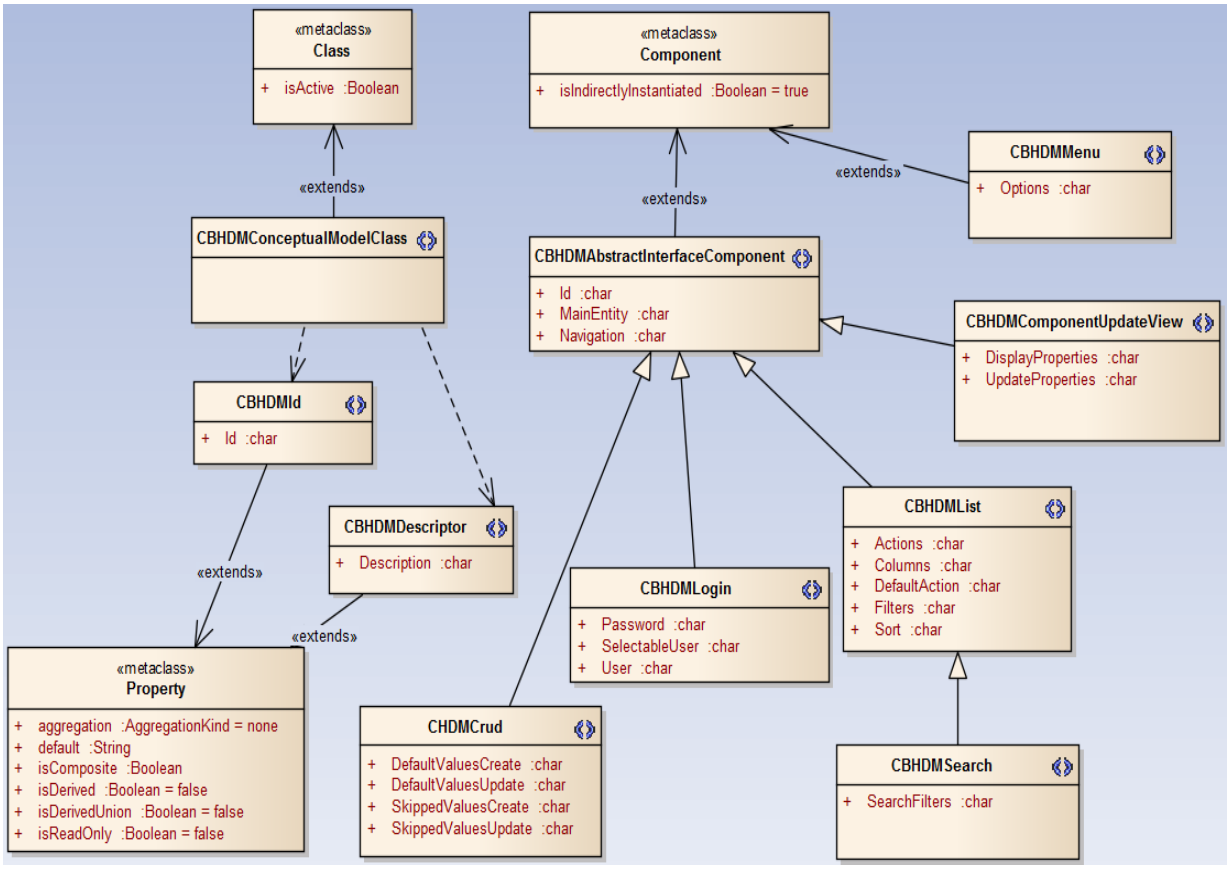


Figura 38. Perfil UML de la metodología CBHDM

3.3.2.2. Generación de Estereotipos

Para construir un profile será necesario determinar el metamodelado del dominio de la aplicación. Si no existe, entonces es necesario definirlo utilizando los mecanismos de extensibilidad provistos por UML. Para lo cual habrá que incluir la definición de las entidades propias del dominio (estereotipos), las relaciones entre ellas, así como las restricciones que limitan el uso de estas entidades y de sus relaciones. A continuación se presentan los estereotipos propios del dominio.

Un estereotipo debe ser creado por cada elemento del metamodelado. Es conveniente que los estereotipos tengan el mismo nombre que los elementos del metamodelado. Una vez identificados los elementos para el metamodelado se realiza la definición de los estereotipos para cada elemento que se está extendiendo.

Es importante tener en cuenta que elementos del metamodelado de UML se están extendiendo y sobre los que es posible aplicar un estereotipo. Ejemplos de tales son

clases, asociaciones, relaciones, operaciones, atributos, etc. De esta forma el estereotipo se aplicara a una metaclase de UML.

3.3.2.2.1. Estereotipos propios y valores etiquetados para el modelo presentado.

- **Diseño conceptual**
  - ***ConceptualModelClass*** *(Estereotipo):* Este estereotipo es utilizado al momento de realizar el diseño conceptual, la cual será una extensión del estereotipo de UML "Clase". De las propiedades de la clase se deberán diferenciar dos atributos fundamentales: el identificador único del objeto y el descriptor.
  - ***Identifier*** *(Estereotipo):* El identificador se corresponde con una clave única que representa unívocamente esa instancia de clase o registro de base de datos al llevarlo a un modelo relacional. Este identificador servirá también para mapear las relaciones entre las tablas de la base de datos. Identifier posee el valor etiquetado Id.
  - ***Descriptor*** *(Estereotipo):* El descriptor será el campo que brinde una descripción amigable para el usuario para identificar el objeto por ejemplo el nombre de una persona, el título de una película, etc. Descriptor posee el valor etiquetado Description.

- **Diseño de Navegación de la interfaz abstracta**
  - ***AbstractInterfaceComponent*** (Estereotipo): Para esta etapa del modelado se utiliza el diagrama de componentes de UML. Para lo cual se definen una serie de componentes estereotipados y de valores etiquetados sobre cada uno de ellos, que permite configurarlos para adaptarlos a las distintas necesidades del sistema. Todos los componentes posen un valor etiquetado común MainEntity que permite especificar la entidad principal de acción del componente.
  - ***Id (Id de componente)*** (Valor Etiquetado): Es un texto que identifica unívocamente al componente y que servirá para poder referirse a él desde otros componentes; por ejemplo al momento de diseñar la navegación.
  - ***Navigation*** (Navegación del componente) (Valor Etiquetado): Se refiere a la barra de navegación presente en cada uno de los componentes o páginas del sistema. Es importante considerar que al estar trabajando en

un ambiente móvil la navegación debe ser simple y estar presente en todas las páginas. En base a las indicaciones sobre navegación en las buenas prácticas del W3C se incorpora la configuración de la navegación a cada uno de los componentes definidos. Lo cual permitirá que el sitio generado a partir del modelo contenga una barra de navegación con links ubicada en la parte superior de la página

Estereotipos Generados:

- **Estereotipo:** Login, Valores etiquetados: MainEntity, User y Password:

  Representa un componente para autenticar el usuario en el sistema. Sus valores etiquetados son: MainEntity (entidad que representa los usuarios), User y Password donde los dos últimos especifican las propiedades la clase de dominio sobre la cual se realizará el chequeo de las credenciales de acceso. El usuario autenticado podrá acceder desde el resto de los componentes del sistema mediante la variable especial llamada LOGGEDUSER.

- **Estereotipo:** List, Valores etiquetados: MainEntity, Filters, Columns y Sort:
  - List: Representa un listado de información que puede verse como una grilla o tabla formada por filas y columnas que muestran información de las entidades del sistema. Sus valores etiquetados son: MainEntity,Filters, Columns y Sort.
  - Filter: representa los filtros que deberán aplicarse para obtener el listado. Estos filtros se realizan sobre propiedades de la entidad principal pudiendo acceder a clases relacionadas mediante notación de objetos.
  - Columns: representa cada una de las columnas que se mostrará en el listado, se refiere a propiedades de la entidad principal y sus entidades relacionadas, pero adicionalmente se le podrán aplicar funciones para generar datos calculados olinks a otros componentes. Las funciones disponibles son: Sum, Count, Exist, Not Exist y Eval. También las columnas podrán contener links a otros componentes o a páginas externas.
  - Sort: indica las propiedades por las cuales estará ordenado el listado

- **Estereotipo**: Search, Valores etiquetados: MainEntity, SearchFilters, FixedFilters, Columns y Sort
  - Search: este componente es similar a List con la diferencia de poder agregar filtros, que en lugar de estar fijos, son ingresados por el usuario al utilizar la aplicación, permitiendo realizar una búsqueda sobre los datos. El resultado de dicha búsqueda será un listado de información. Sus valores etiquetados son: MainEntity, SearchFilters, FixedFilters, Columns y Sort. Tanto Columns como Sort son idénticos a los definidos en el componente List y el valor etiquetado FixedFilters es idéntico a Filter pero se renombra para mayor claridad. El único valor etiquetado adicional es SearchFilters, que incorpora los filtros de búsqueda ingresados por el usuario. En este se definen las propiedades sobre las cuales se realizará la búsqueda pudiendo especificar para cada una de ellas un tipo de filtro siendo los valores posibles: SingleSelection, MultipleSelection, FreeText y BooleanType.
  - SingleSelection: representa un filtro de selección simple en el cual el usuario podrá seleccionar un único valor.
  - MultipleSelection representa un filtro de selección múltiple en el cual el usuario podrá seleccionar uno ó más valores.
  - FreeText representa un filtro de texto libre.
  - BooleanType representa un filtro que puede tomar sólo dos valores verdadero o falso

- **Estereotipo**: Menu, Valor etiquetado: option

  Es un componente que representa un menú de links a los distintos componentes. Contiene el valor etiquetado Options que permite definir los distintos links que conformarán las opciones del menú a mostrar al usuario.

- **Esterotipo:** CRUD, Valores etiquetados: MainActivity, DefaultValuesCreate, SkippedPropertiesCreate, DefaultValuesUpdate, SkippedPropertiesUpdate

  Este componente es el encargado del crear, mostrar, actualizar ó eliminar un objeto de una clase.

  Sus valores etiquetados son: MainEntity,DefaultValuesCreate, SkippedPropertiesCreate, DefaultValuesUpdate, SkippedPropertiesUpdate.

  Si hay ciertos datos que se deben completar en el objeto pero no se solicitan al usuario se utilizan DefaultValuesCreate ó DefaultValuesUpdate, lo que

permite por ejemplo completar de forma automática el usuario que creó el registro y la fecha y hora de modificación. SkippedPropertiesCreate ó SkippedPropertiesUpdate son aquellas propiedades que deben quedar intactas, por ejemplo: al crear un registro no se completa el usuario de modificación; al modificar un registro la fecha de creación queda intacta. El comportamiento del componente estará dado por un parámetro que contendrá el link que provoque la activación del mismo. Ese link deberá contener el parámetro Action siendo los valores posibles C, R, U y D indicando cada letra, cada una de las acciones posibles: C (crear), R (mostrar), U (actualizar) y D (eliminar). Adicionalmente este componente, para las operaciones de modificación y eliminación, debe recibir el parámetro ObjectID que va a indicar la clave primaria del objeto sobre el cual se va a llevar a cabo la acción.

- **Estereotipo:** ComponentUpdateView, Valor etiquetado: (DisplayProperties y UpdateProperties): Es el componente de presentar vistas que permiten la actualización de determinados campos a través de una grilla o de un formulario de edición.

3.3.2.3. Análisis de las Clases de las que derivan los nuevos estereotipos.

Asociación entre los elementos de extensión y las metaclases

En la tabla 9 se puede observar cada elemento del profile asociado a un estereotipo determinado y la meta clase de UML a la que pertenece o del cual deriva el estereotipo generado. Al momento de generar un profile se debe hacer una clasificación de los estereotipos que lo van a componer e indicando la metaclase del conjunto de UML al que pertenecen o derivan como se detalla en la tabla 9.

Tabla 9. Elementos del Profile

Elementos del Profile	Estereotipo	UML Metaclass
ConceptualModelClass	CBHDMConceptualModelClass	Class
Identifier	CBHDMId	Property
Descriptor	CBHDMDescriptor	Property
AbstractInterfaceComponent	CBHDMAbstractInterfaceComponent	Component
Login	CBHDMLogin	CBHDMAbstractInterfaceComponent
List	CBHDMList	CBHDMAbstractInterfaceComponent
Search	CBHDMSearch	CBHDMList
Menu	CBHDMMenu	Component
Crud	CBHDMCrud	CBHDMAbstractInterfaceComponent
ComponentUpdateView	CBHDMComponentUpdateView	CBHDMAbstractInterfaceComponent

3.3.2.4. Creación del MOF (Meta Object Facility)

Primeramente se debe plantear se va a crear un lenguaje nuevo ó bien un lenguaje basado en UML. Si se trata de un lenguaje nuevo deberá definirse a través del MOF (Meta Object Facility) [OMG13] de hecho UML tiene su definición a través de un MOF. En el caso de tratarse de un lenguaje basado en UML deberá analizarse que cambios tendrá sobre UML si dichos cambios no son conservativos (cambian la semántica de UML) también deberá realizarse un MOF. Ahora bien en el caso del presente proyecto para el lenguaje a generar bastará con una extensión conservativa del UML. Es por ello que no resulta necesario volver a construir un MOF y se decide realizar un profile (perfil). Existe una presentación interesante accesible en la página de la OMG, en donde un autor [DEF00] plantea el debate sobre (a) Constuir un Perfil de UML; ó (b) Trabajar para extender un MOF.

Los perfiles permiten agregar nuevos elementos a través de estereotipos a los que incluso pueden asignárseles construcciones gráficas, pero sin modificar la semántica del lenguaje.

3.3.2.5. Modelado de un caso de prueba.

A continuación se mostrará la selección del software y el modelo del primer caso de ejemplo.

3.3.2.6. Selección de Software para el modelado.

Según la metodología con la que se realice el modelado cambiará inevitablemente el software a seleccionar. Por ejemplo, si se modelará en IFML se necesitará WebRatio (cabe recordar que IFML surge este año sustituyendo a WebML). Incluso algunas metodologías no cuentan con herramientas disponibles para su modelado como es el caso de OOHDM, esto implica que no se cuenta con una herramienta que permita tener las construcciones gráficas propias del lenguaje.

En este apartado se presentan ejemplos con su modelado correspondiente. Estos ejemplos son propios del dominio en cuestión, presentados primeramente con su explicación correspondiente, generando el diagrama de clases con el EA (Enterprise Architect). Luego se decide modelar con OOHDM a pesar de no contar con una herramienta específica para realizarlo.

3.3.2.7. Diseño del primer caso de ejemplo.

Caso Ejemplo: Registración de Tareas.

Es 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.

- **Diagrama de Clases**

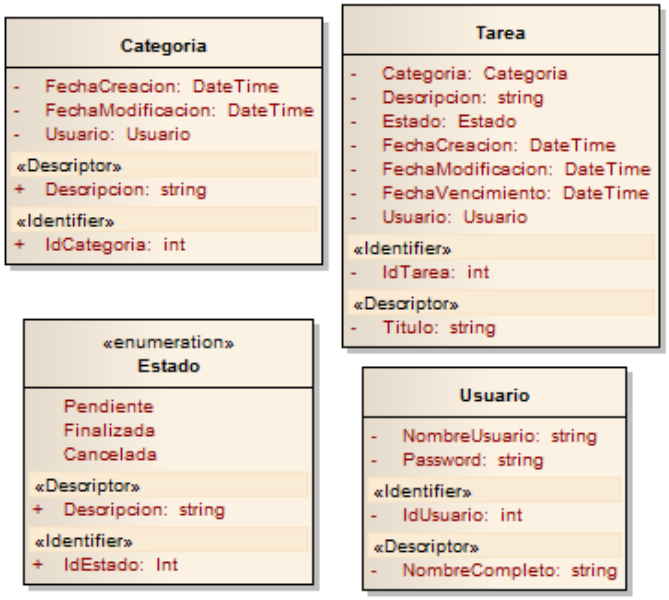


Figura 39. Diagrama de Clases Ejemplo 1.

- **Propiedades especiales para auditoría:**
  Algunas clases incorporan propiedades especiales con información para auditoria. Estas propiedades son completas en forma automática por el sistema de la siguiente forma:
  - 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
  - Usuario: se completa con el usuario de creación del objeto

- **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:
  - Ver 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 hacer clic en la descripción el usuario será redireccionado a la pantalla de edición de dicha categoría. Mediante una presión larga sobre una fila o mediante un botón secundario se debe acceder a un menú de acciones donde el usuario contará con la opción de eliminar dicha categoría (para eliminar lo llevará a la pantalla de edición donde verá los datos y tendrá un botón de eliminar)

- **Edición de Categoría:** Esta pantalla permite realizar el alta, baja y modificación de una categoría. El usuario solo ingresa la descripción de la misma.

- **Listado de 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”.

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

  El listado deberá tener las siguientes columnas:
  - Titulo
  - Categoría
  - Fecha de Vencimiento

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

- **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 si 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)**

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: selección simple**
  - **Descripción: texto libre**
  - **Categoría: selección simple**

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

- **Titulo de la tarea**
- **Categoría**
- **Estado**
- **Fecha de Vencimiento**

Por defecto el listado solo mostrará tareas del usuario logueado.

3.3.2.8. Modelado del ejemplo: “Registración de Tareas” – Modelado en OOHDM

- **Diseño Conceptual.**

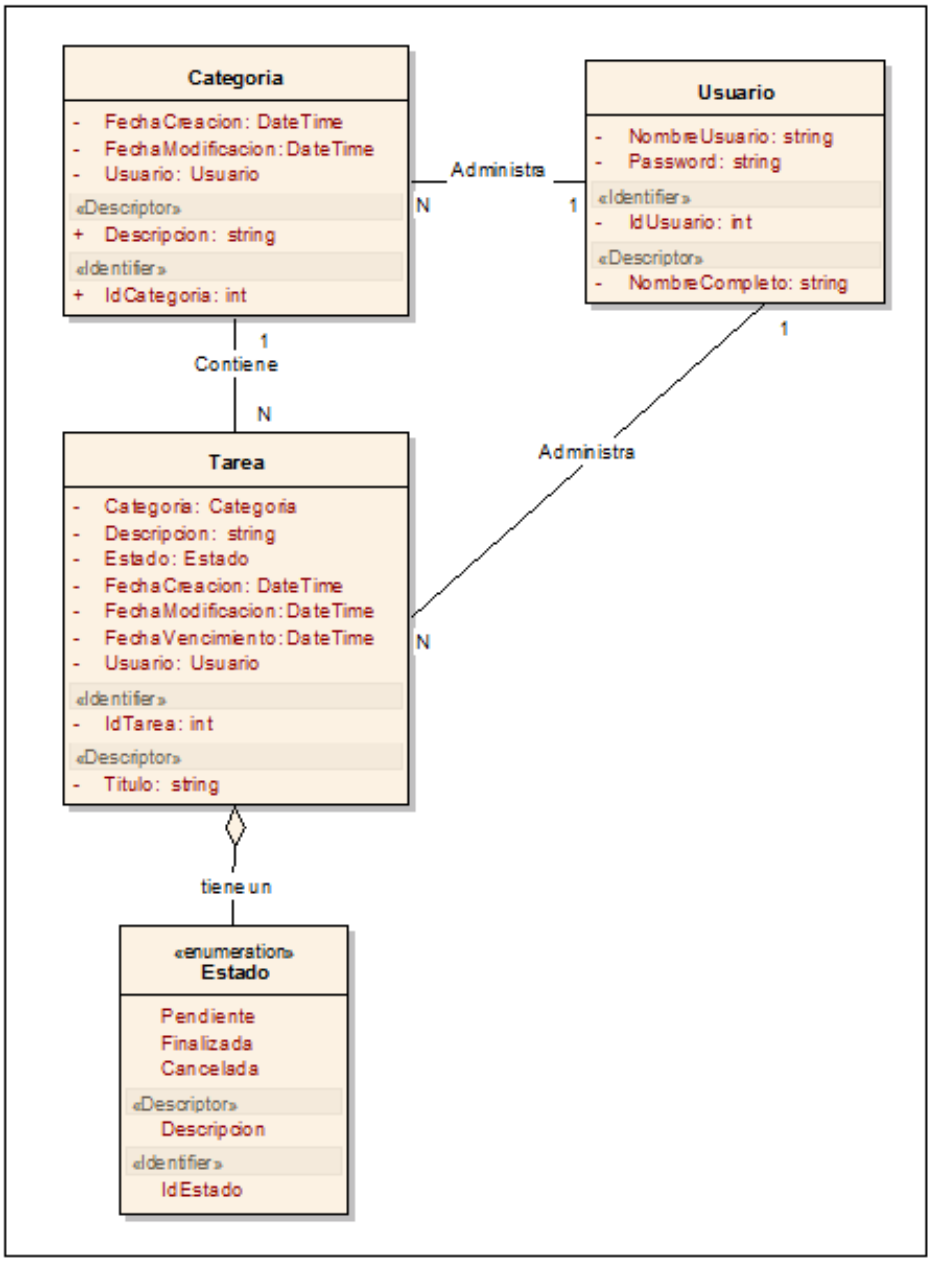


Figura 40. Modelo Conceptual del Sistema de "Registración de Tareas"

- **Diseño Navegacional.**

  - ✓ **Esquema Navegacional**

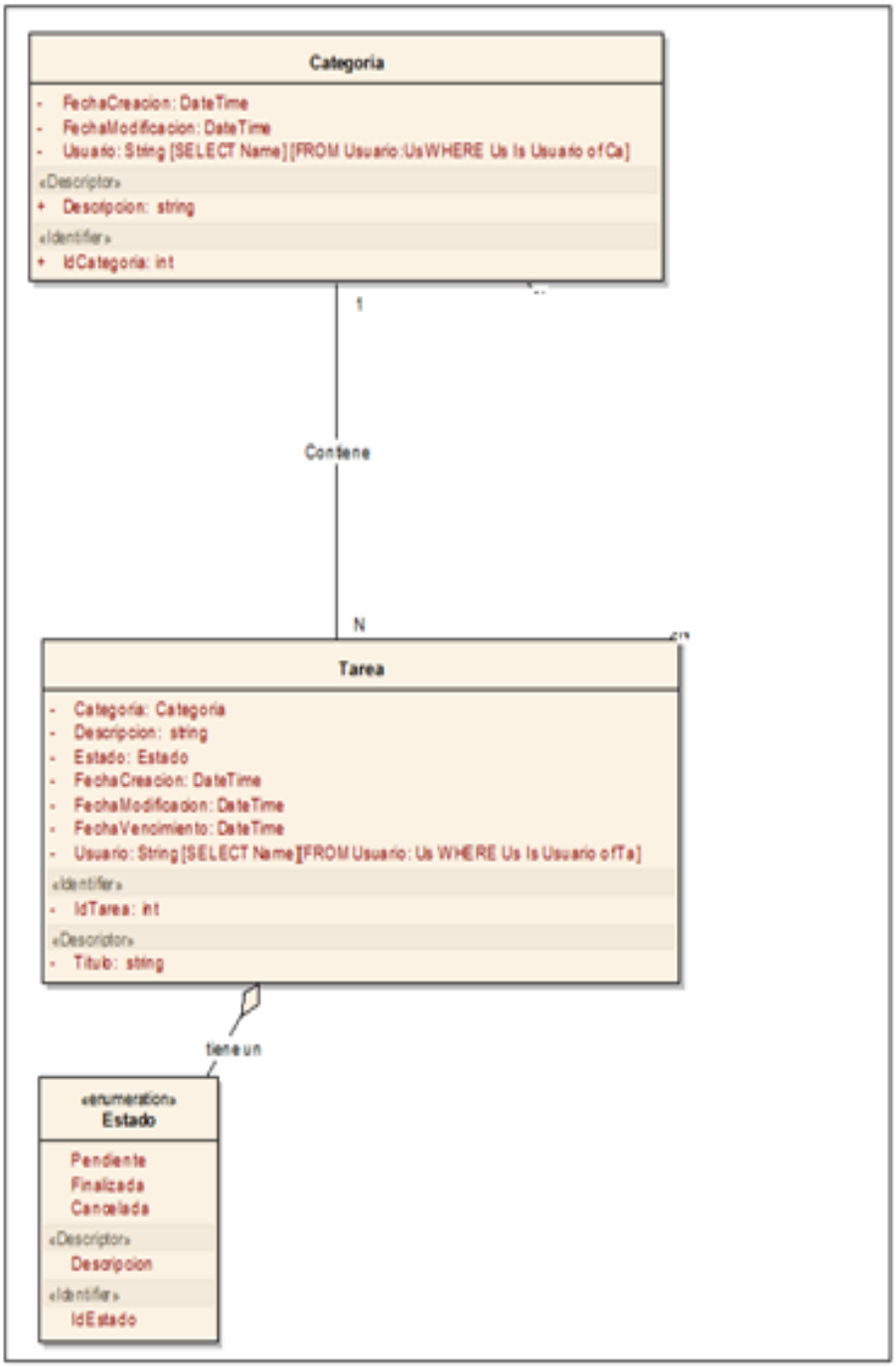


Figura 41. Esquema Navegacional del Sistema de "Registración de Tareas".

✓ **Esquema de Contextos Navegacionales.**

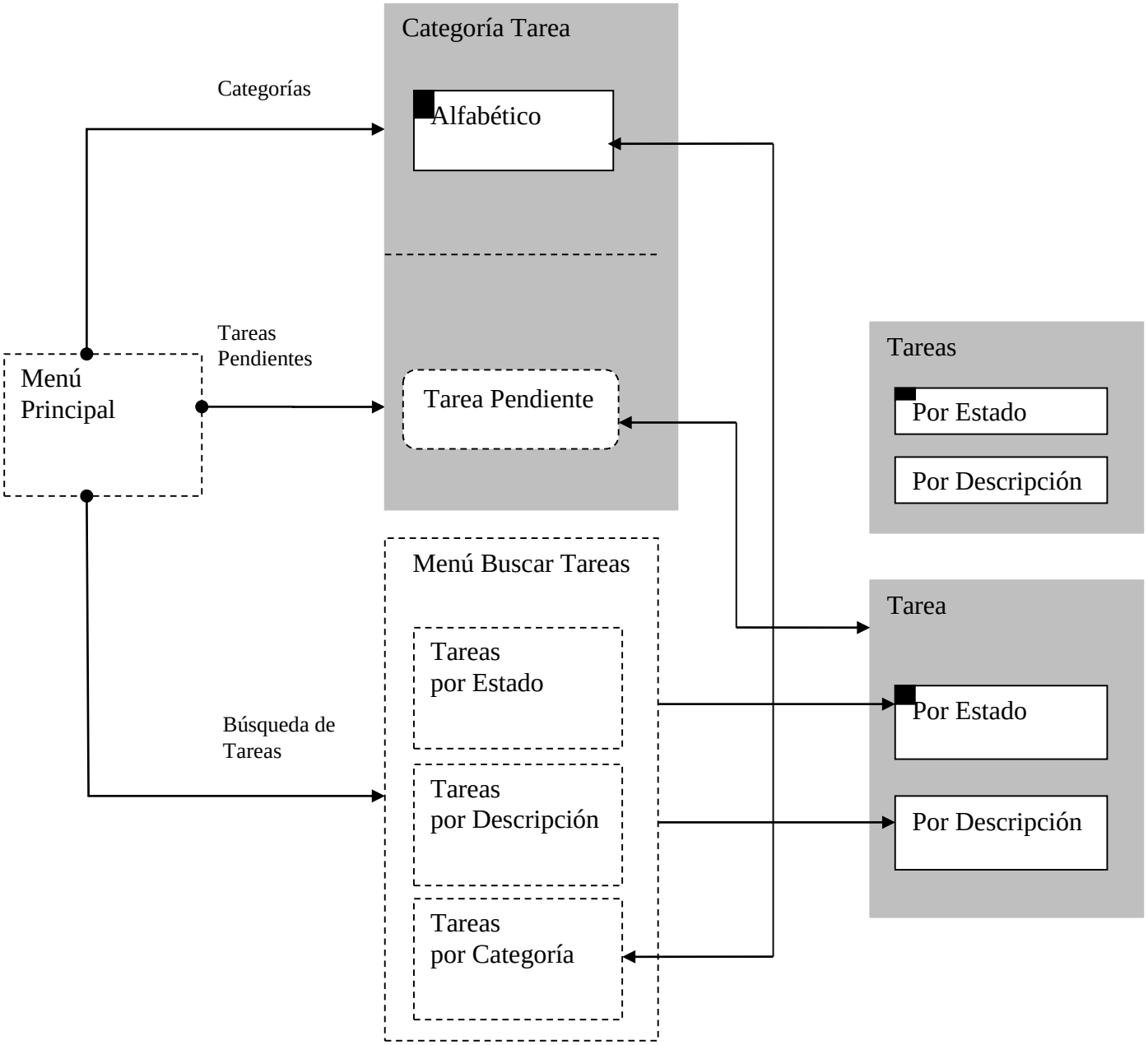


Figura 42. Esquema de Contextos Navegacionales.

- **Diseño de Interfaces Abstractas.**

✓ **ADV Tarea.**

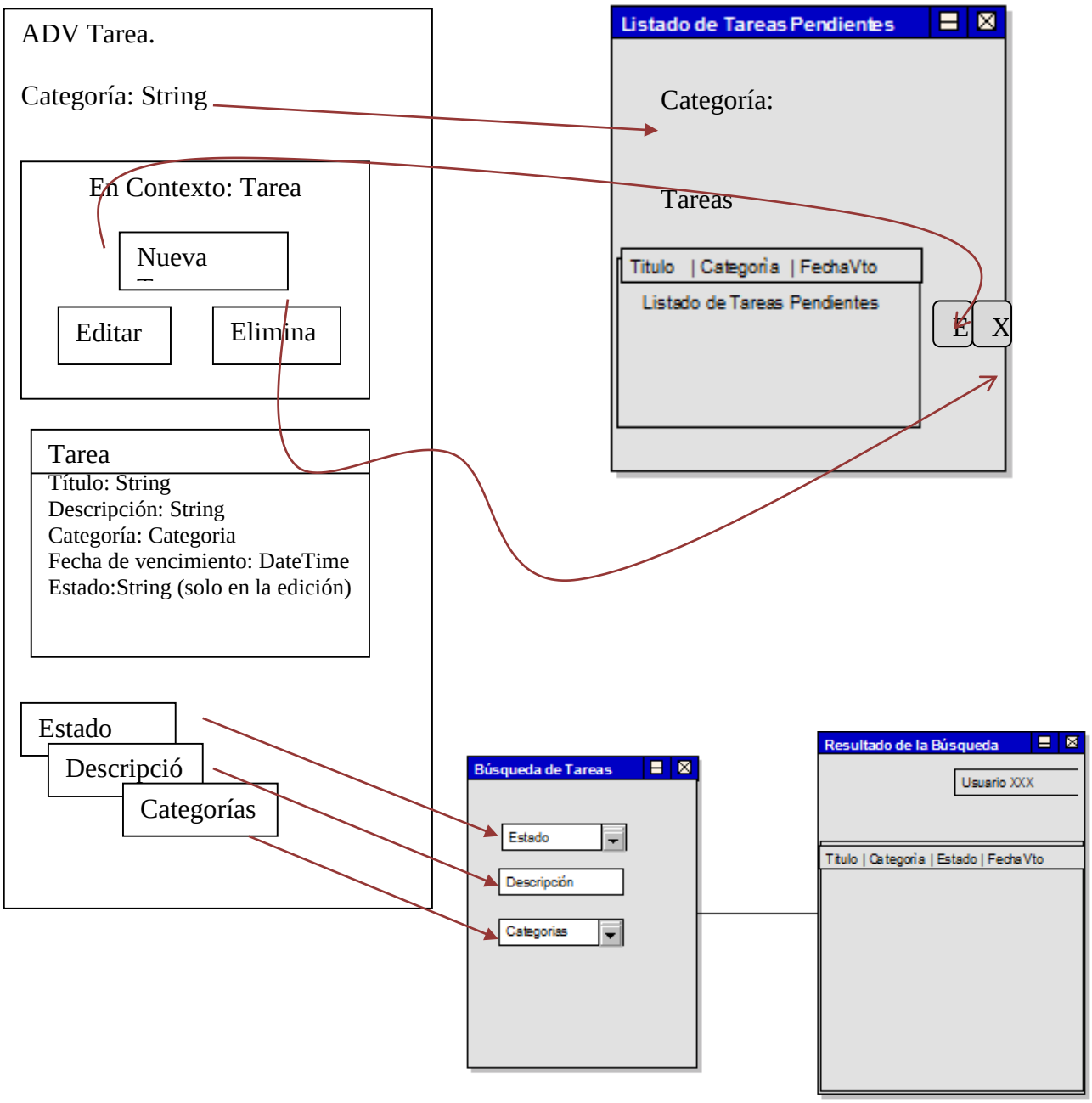


Figura 43. "ADV Tarea" y su relación con los Objetos de Interface.

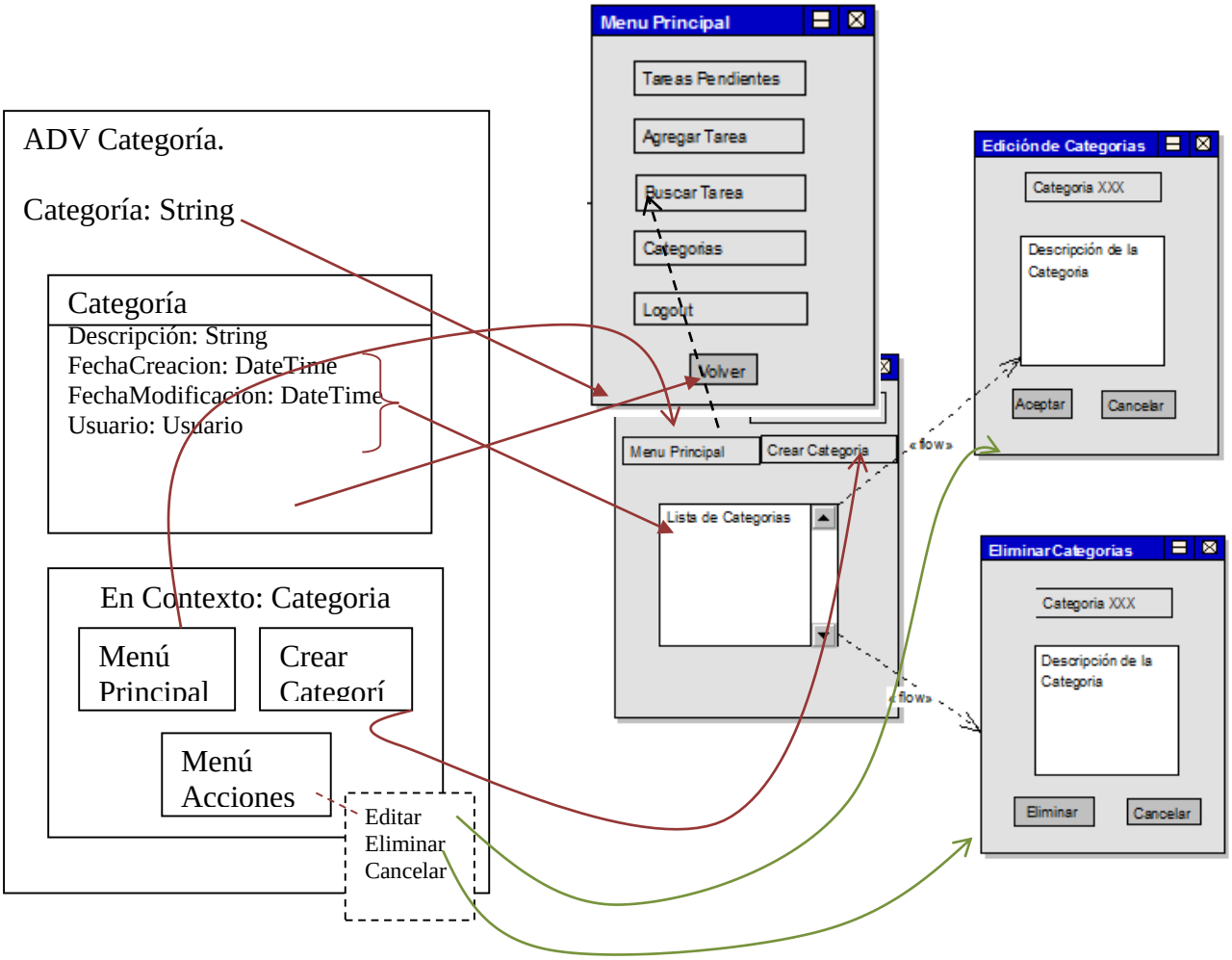


Figura 44. ADV Categoría y su relación con los Objetos de Interface.

Implementación.

En ésta etapa se deberá realizar la implementación ó puesta en marcha de la aplicación.

3.3.2.9. Modelado del ejemplo: “Registración de Tareas” – Modelado en IFML

El modelado de éste caso se realizo con WebRatio Community Platform Version: 7.2.7

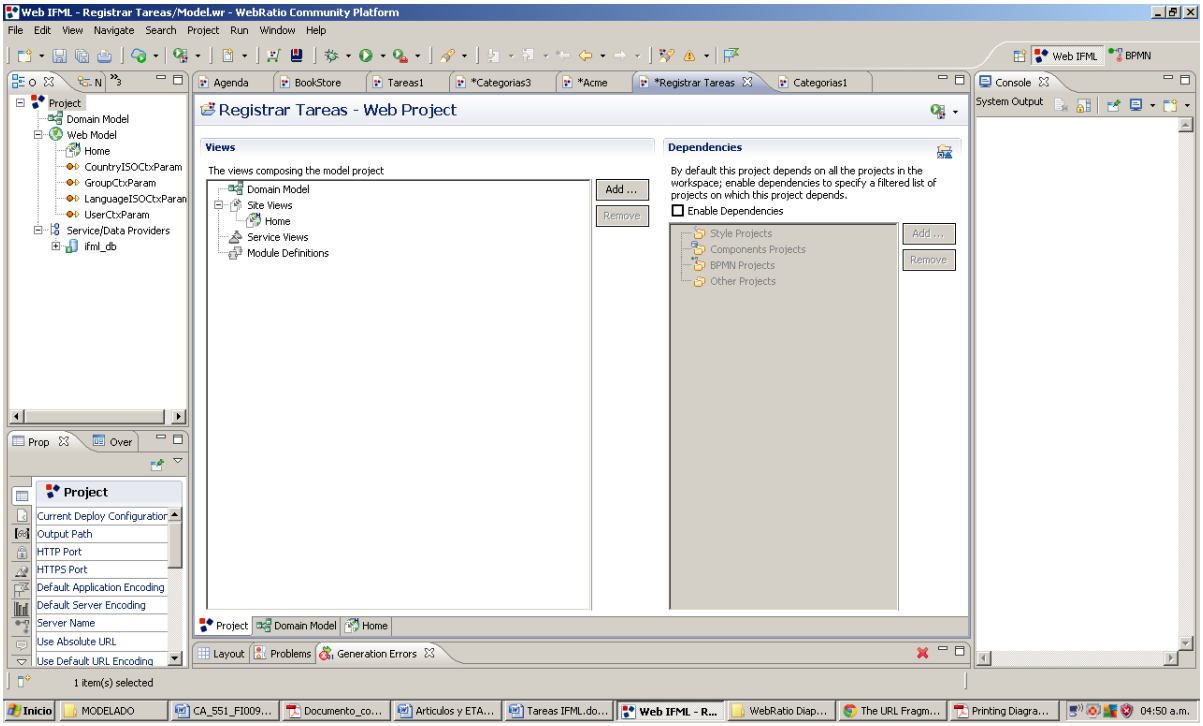


Figura 45 Pantalla de WebRatio. Vista Proyecto.

- **Modelo de Dominio.**

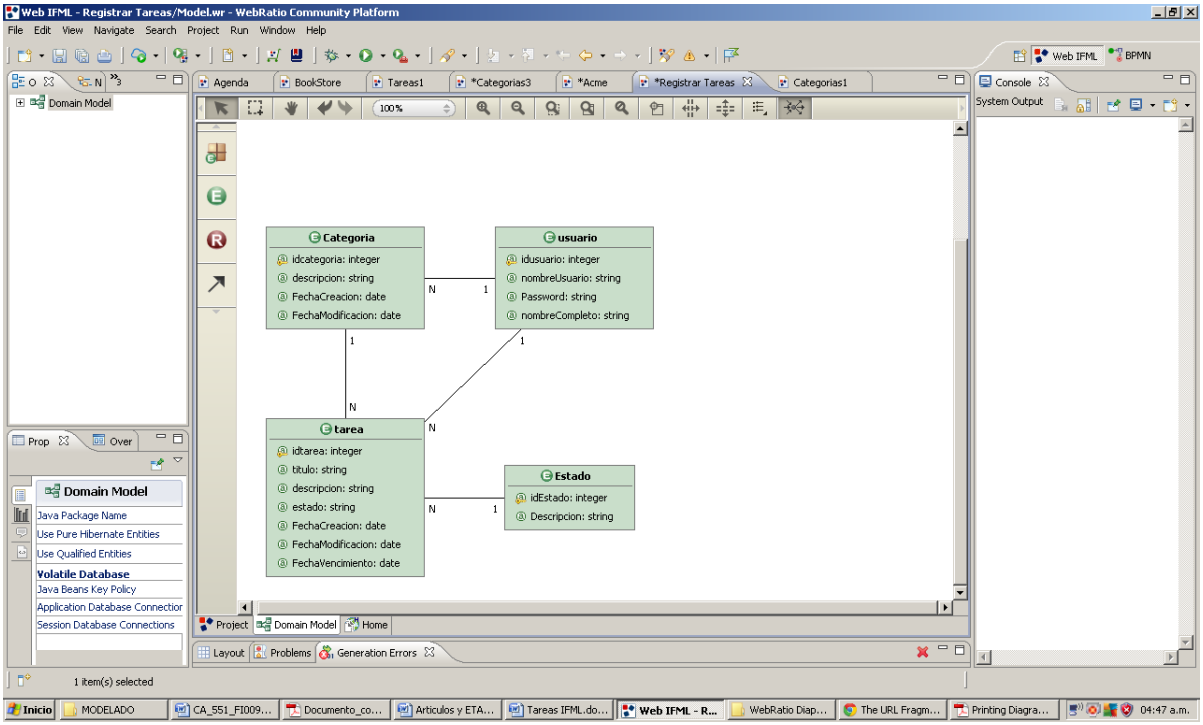


Figura 46. Pantalla de WebRatio. Vista Modelo de Dominio.

Para el Modelo de Dominio, podría utilizarse como Administrador de Base de Datos, DB2, Microsoft SqlServer, MySql. Este último es el que se utilizo en el ejemplo.

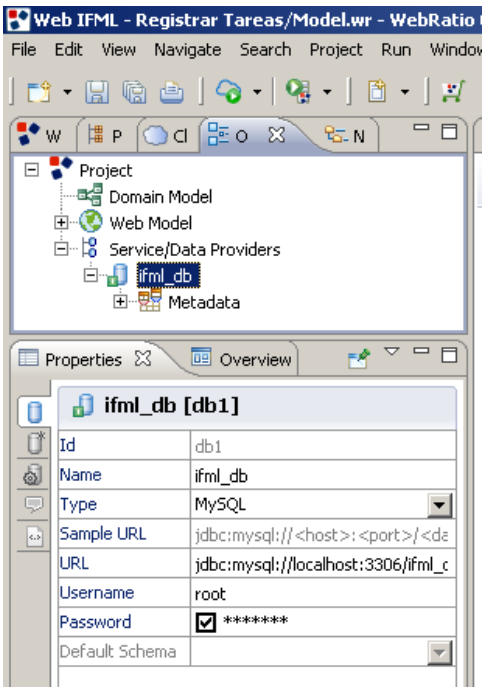


Figura 47. Ventana de Propiedades del Servicio de Datos.

- **Vista de Sitios – Home.**

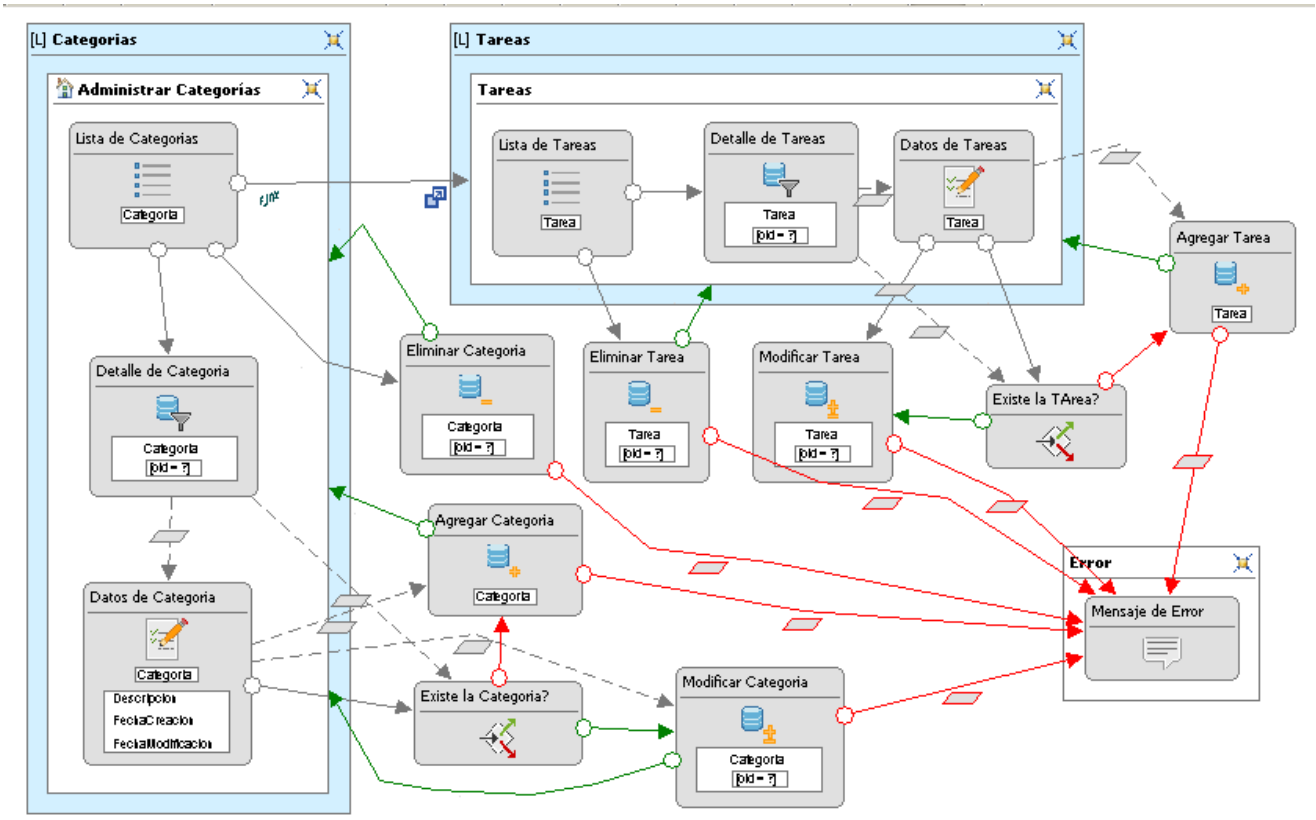


Figura 48. Diagrama IFML. Vista de Sitios.

3.3.3. Etapa 3. Refinado del Modelado

Esta refinación se logrará modelando distintas aplicaciones las cuales incrementen gradualmente el grado de complejidad, pudiéndose encontrar nuevas necesidades. La generación de Código Fuente se efectuará utilizando una herramienta a construir, generada mediante OO (Orientación a Objetos) incluyendo la utilización de template que permitirá la obtención de código fuente en distintos lenguajes.

La figura 49 muestra el esquema general de la metodología planificada.

Figura 49. Esquema general de la Metodología Planificada

3.3.3.1. Modelado de Caso de Ejemplo con Distintas Herramientas

(A) Caso Ejemplo: "Sistema Móvil para administrar viajes de Taxis"

Sistema Móvil para administrar viajes de taxi donde los taxistas pueden ver y aceptar los viajes disponibles. Un chofer puede aceptar más de un viaje al mismo tiempo por razones de proximidad pero solo puede iniciar un viaje a la vez.

La aplicación tiene un fronted móvil y sitio web estándar de backend para la administración. El administrador será el encargado de agregar los nuevos viajes al sistema y los choferes accederán a esos viajes desde sus dispositivos móviles.

Diagrama de clases:

Figura 50. Diagrama de Clases del ejemplo 2. Viajes

Pantallas del Sistema de Backend de Administración:

Login: Pantalla de ingreso al sistema y autenticación del usuario administrador

Menú: El menú principal contendrá las siguientes opciones:

- Viajes Actuales
- Viajes Históricos
- Choferes
- Administradores
- Logout

Viajes Actuales: muestra un listado de los viajes actualmente en curso. Es decir son aquellos objetos del tipo Trip cuyo estado es Pending, Accepted o Started. Se deberá mostrar una grilla con las siguientes columnas:

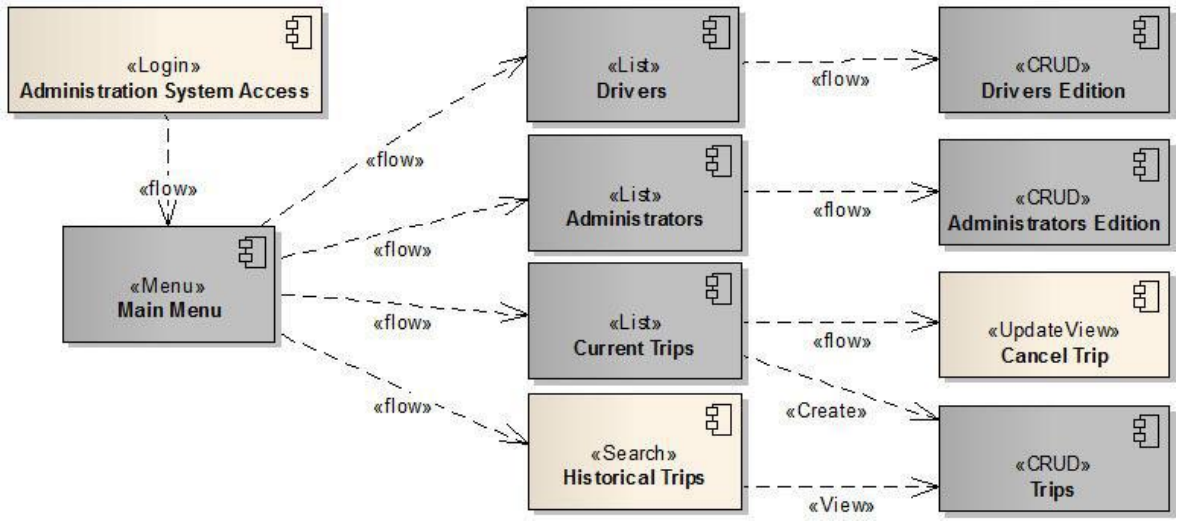


Figura 51. Pantallas del Usuario Administrador.

Pantallas del Sistema Móvil (Frontend)

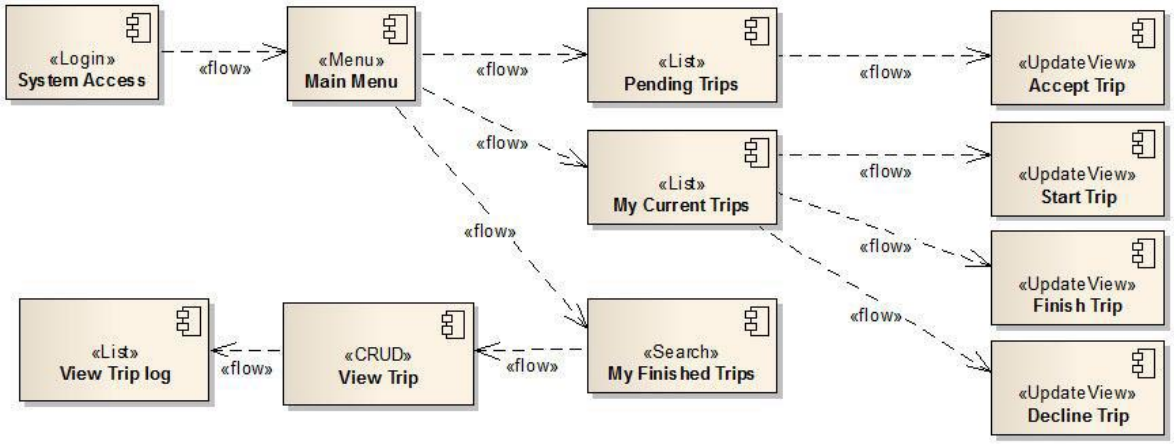


Figura 52. Pantallas del sistema Movil

(B) Modelado del Caso Ejemplo: "Sistema Móvil para administrar viajes de Taxis".

Modelo Conceptual

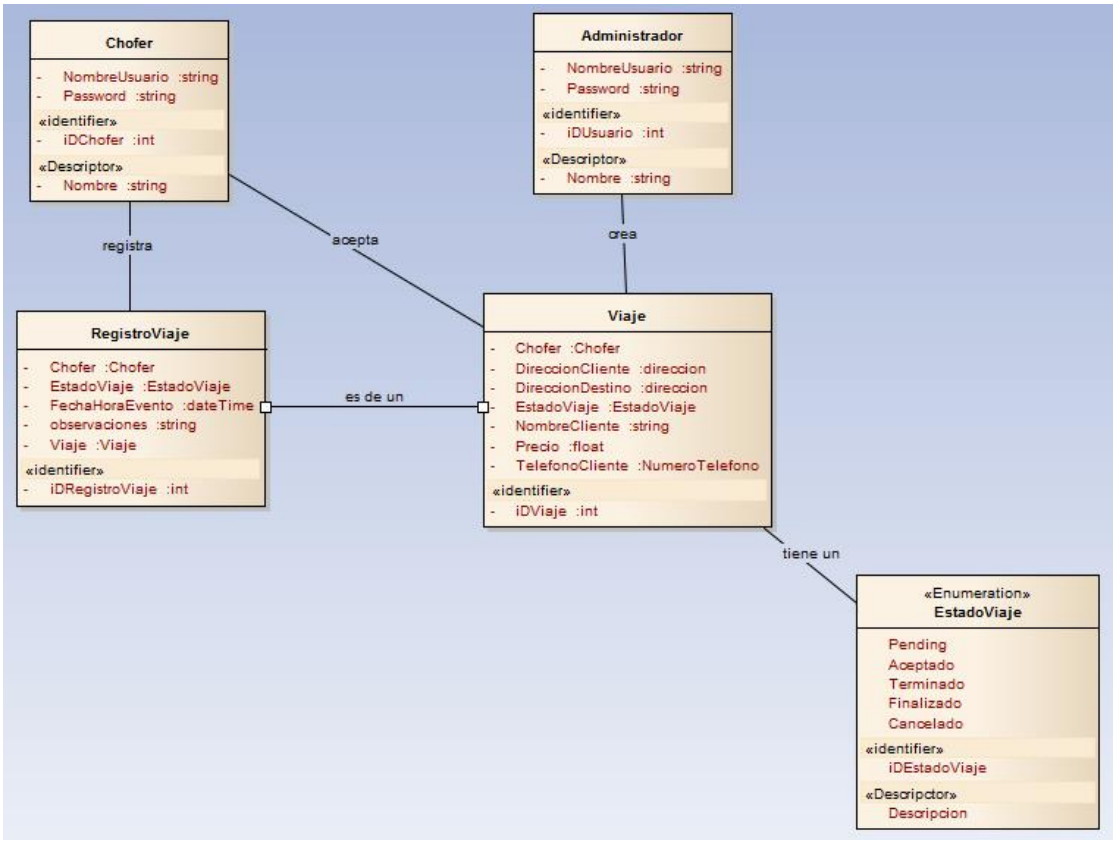


Figura 53. Pantallas del sistema Móvil

Modelo Navegacional

Esquema de Navegación: Taxis

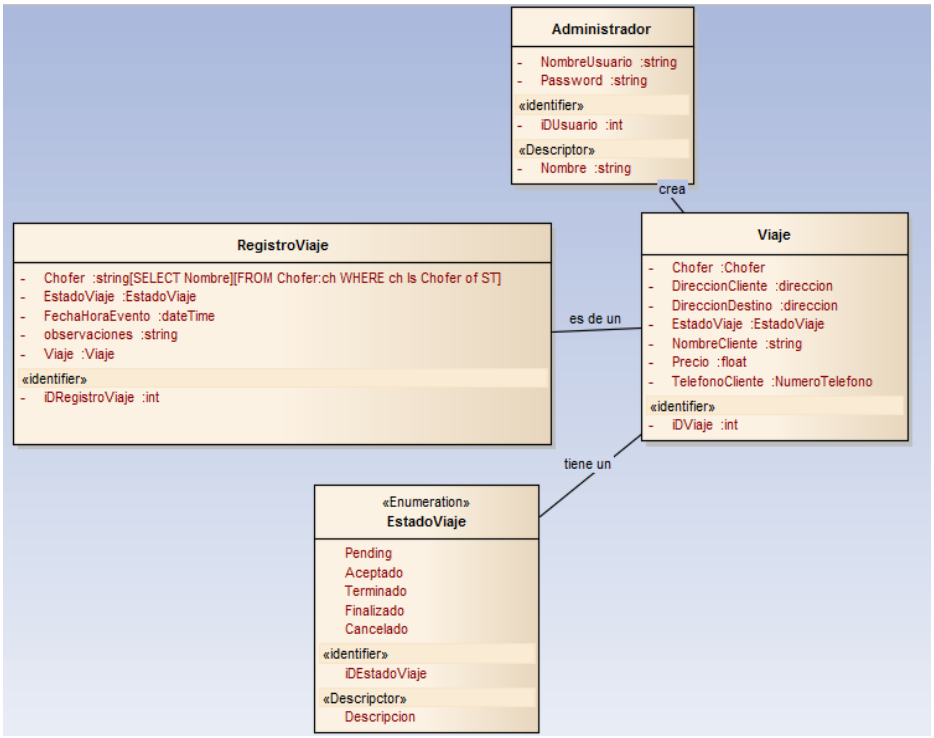


Figura 54. Esquema de Navegación. Taxis.

Esquema de Contexto Navegacional para el **Usuario Móvil.**

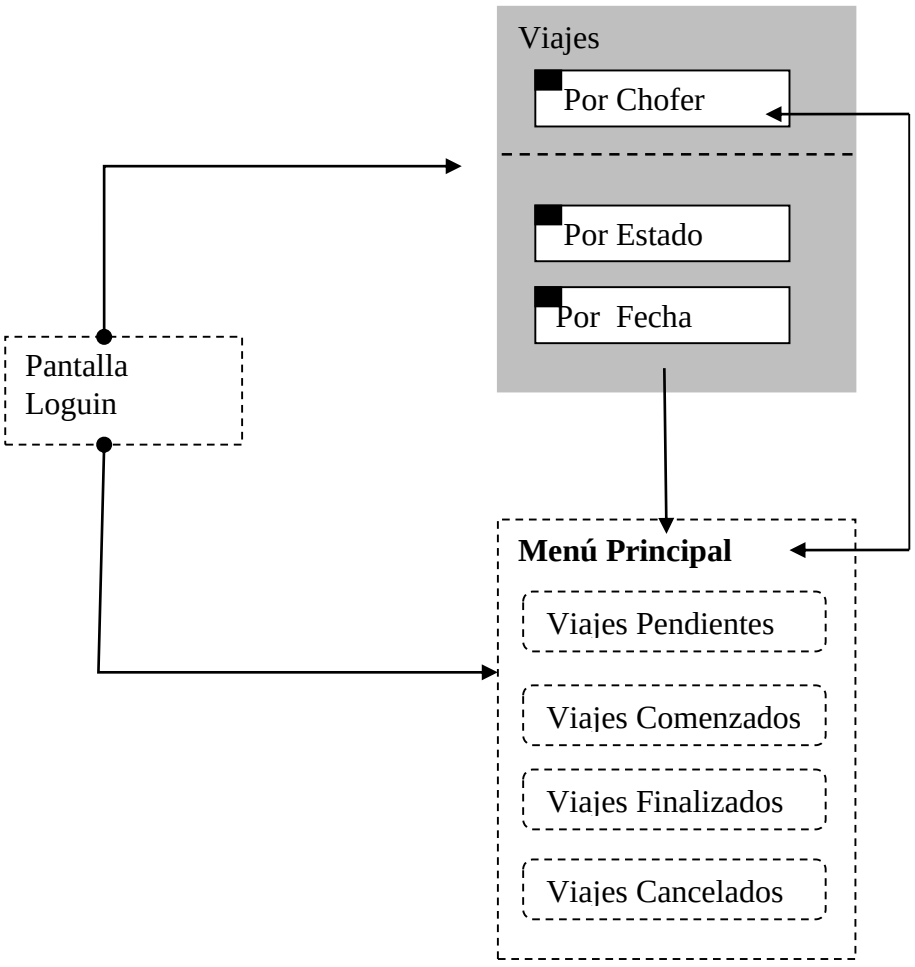


Figura 55. Esquema de Contexto Navegacional para el usuario movil.

Esquema de Contexto Navegacional para el **Usuario Administrador.**

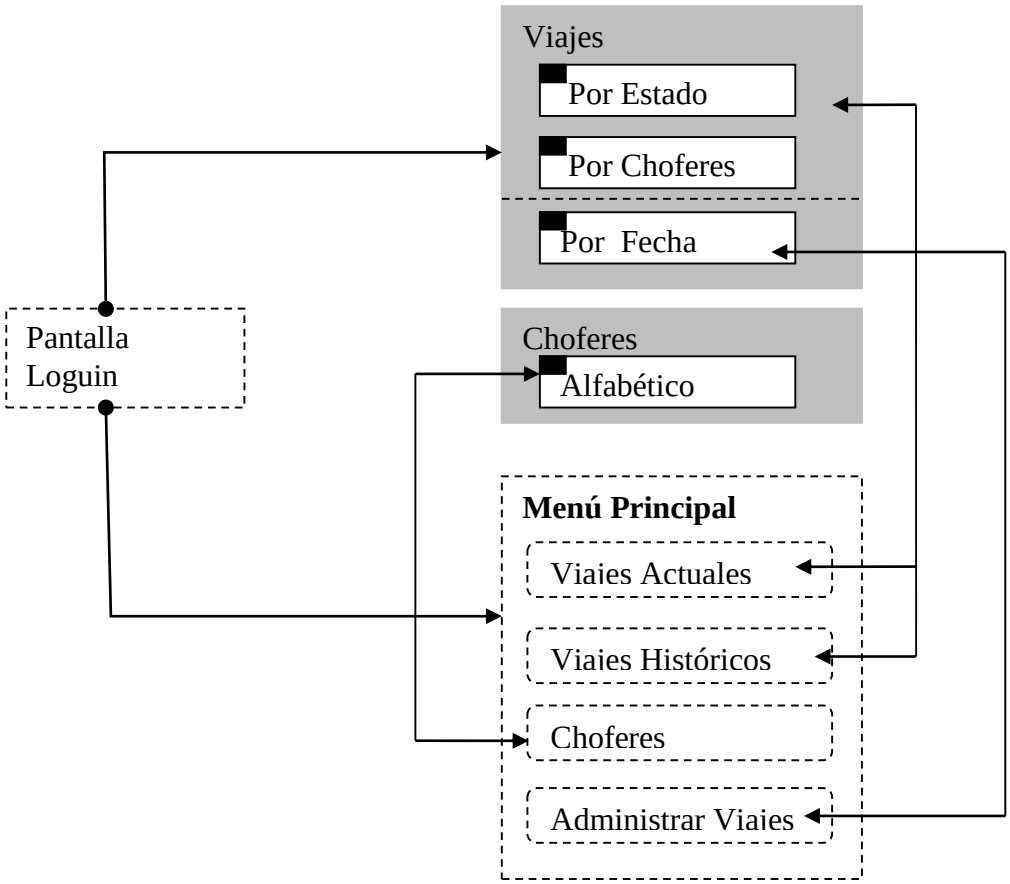


Figura 56. Esquema de Contexto navegacional para el usuario administrador.

Diseño de Interfaces Abstractas.

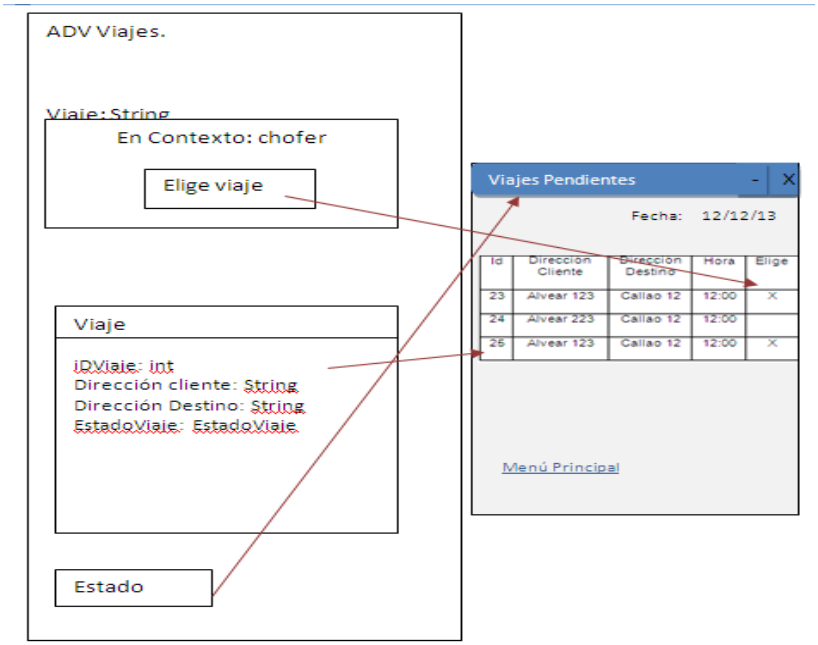


Figura 57. ADV Viajes.

(C) Roles de Usuarios.[2]

Muchas aplicaciones muestran y ocultan funcionalidades a los usuarios según el rol que poseen. Se entiende por rol a una función particular dentro del sistema que lo habilita para realizar determinadas tareas. Por ejemplo un administrador será el responsable de configurar el sistema, mientras que otros tipos de usuarios no deben poder modificar dicha configuración.

El modelo permite definir distintas funcionalidades de acuerdo al rol del usuario logueado, para ello se agrega un parámetro adicional a la función de links: RoleCondition.

Este parámetro agrega una regla que debe ser chequeada para determinar si el link será visible o no para el usuario logueado. Quedando la forma de la función link de la siguiente manera:

```
<Link>::='Link('<LinkText>','<BrowsableComponent>','
<OptionalLinkParameters>','<OptionalAccessKey>','
<OptionalRoleCondition> ')'
```

La condición se relaciona en forma automática al usuario logueado, por lo tanto el punto de inicio para chequear las condiciones será la clase que representa a dicho usuario. Este enfoque se adapta a diferentes formas de representar la asignación de seguridad:

- **Información del rol directamente en la clase de usuario mediante campos del tipo boolean.**
- **Un único rol por usuario con una clase relacionada**
- **Múltiples roles de usuario con una clase relacionada**

Con cada uno de estos enfoques solo cambiará la forma en que se debe configurar el parámetro de la condición del Rol, utilizando la notación de objetos como el resto del modelo.

3.3.4. Etapas 4 – Análisis del XMI.

- ✓ **XMI (XML Metadata Interchange)**

  XMI es un formato de intercambio de metadato mediante XML, en el mismo se representan modelos UML, tanto su definición como las modificaciones sobre los mismos.

  El estándar definido por la OMG (Object Management Group) 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

[2] Este tema ha dado origen a un artículo presentado en CACIC 2013.

herramienta, inclusive se permite hacer modificaciones sobre los diagramas y luego exportar dichos cambios.

✓ **Definición**

En el XMI (XML Metadata Interchange) se integran tres estándares:

- XML - eXtensible Markup Language
- UML - Unified Modeling Language
- MOF - Meta Object Facility (Un metamoledado de OMG y un repositorio estándar de metadato)

Teniendo a UML como estándar de modelado de objetos, el estándar MOF que define un extensible framework para definir modelos para metamodelos y proporcionando herramientas con interfaces de programación, para acceder a los metadatos. Ellos junto a XML permiten el intercambio con un formato estándar. En la siguiente imagen se muestra la arquitectura del MOF tomada de [IYE00]

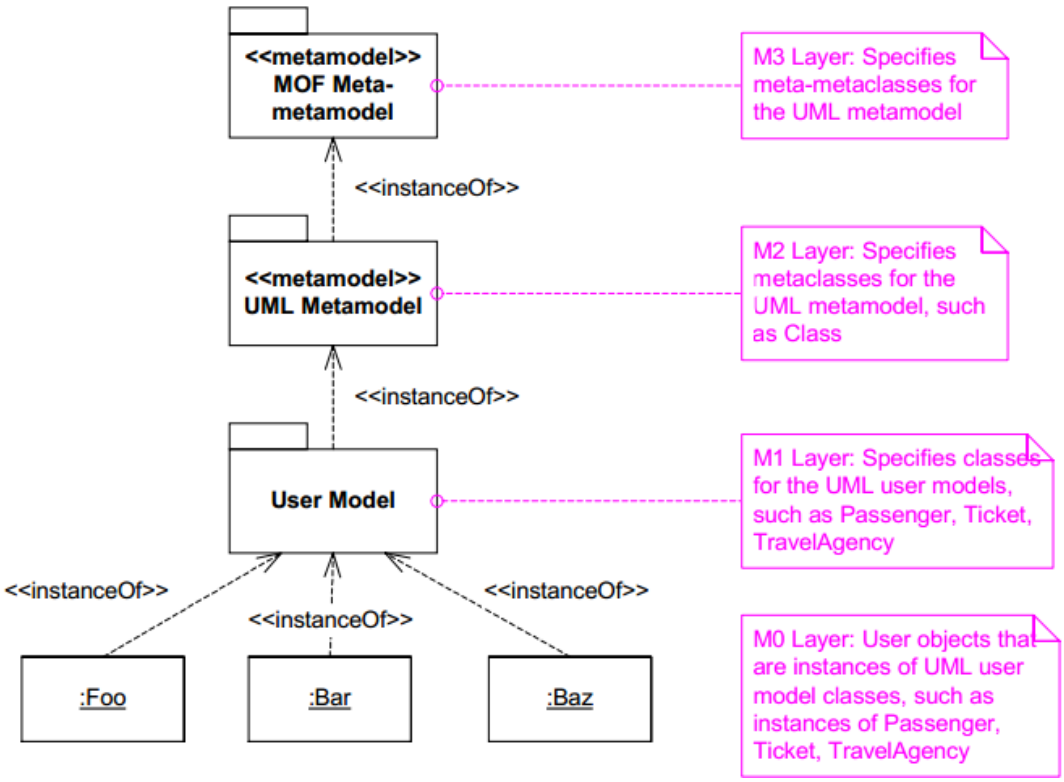


Figura 58. Arquitectura del metamodelo OMG

✓ **Armado del XMI**

Los aspectos necesarios para armar un correcto XMI para un documento básico son:

- Versión del XML por ejemplo <? XML versión ="1.0">

- Opcionalmente se puede agregar una declaración de codificación por ejemplo <? XML version="1.0" ENCODING="UCS-2" ?>
- Algunas instrucciones de procesamiento XML.
- Un esquema de element XML
- Declaraciones para un Metamodelo específico
- Declaraciones para las extensiones

Cada documento XMI consta de las siguientes declaraciones, a menos que el XMI está incrustado en otro documento XML:

- Una instrucción de procesamiento XML versión
- - Una declaración de codificación opcional que especifica el conjunto de caracteres
- - Cualquier otra instrucción de procesamiento XML válidos

✓ **Namespace**

Es sumamente importante la asociación de un namespace en todo documento XMI, ya que la misma hacer que todos los tags definitorios de los distintos elementos cambien sustancialmente entre los distintos documentos. También es importante la asociación con los distintos namespaces, ya que teniendo un mismo diagrama por ejemplo, al asociarlo a distinos namespaces, los XMI serán realmente distintos, por ejemplo si se asocia con el namespace xsd (Schema XML) la declaración de una clase se realizará a través de la utilización del tag "<xsd:complexType name=Clase1" > donde Clase1 indica el nombre de la clase, sin embargo, en el caso que se esté utilizando el namespace de UML, la declaración de las clases se realizarán utilizando el tag "<UML:Class name=Clase1">. Por supuesto se pueden utilizar namespaces propios donde se declaran de diversas formas (como es el caso de ArgoUML).

El siguiente es una asociación con el namespace de UML

```
<xmi:XMI version="2.0"
xmlns:UML="http://schema.omg.org/spec/UML/1.4"
xmlns:xmi="http://schema.omg.org/spec/XMI/2.0">
```

✓ **Utilización de esquemas**

Dentro del estándar se presenta la posibilidad de detallar el esquema, de esta forma uno provee la estructura que se va a utilizar en el XMI, con esto se puede validar la sintaxis y parcialmente la semántica del documento. Según la documentación desarrollada en [OMG05b], sugiere el agregado del esquema para

cuando se procese el documento XML, se pueda comprobar que los elementos requeridos en el esquema se encuentran efectivamente en el desarrollo del mismo. Sin el esquema sólo se podrá validar que esté formateado como un XML estándar. Tomando como referencia la utilización del namespace xsd, la forma de declarar una clase es con el agregado del tag "xsd:complexType" por ejemplo:

`<xsd:complexType name="clase">` dentro se pueden poner los distintos atributos y propiedades de las clases por ejemplo:

```
<xsd:complexType name="persona">
<xsd:element name="name" type="xsd:string"/>
</xsd:complexType >
```

Al ser opcional la declaración de la misma, algunas herramientas case prefieren incluirlo y otras no consideran que sea necesario.

✓ **Información dentro del XMI**

Como el XMI permite exportar e importar, no sólo diagramas sino también la información volcada en ellos. A continuación se muestra el caso de los objetos, atributos y relaciones:

- Los objetos pueden declararse de la siguientes manera [OMG05c]:
  `<Departamento xmi:id="Departamento1"/>`
  Otra forma de representar sería:
  `<complexco:departamento xmi:id=”Departamento1”/>`
- Los atributos pueden completarse de la siguiente manera:
  `<Departamento xmi:id="Departamento1" number=”13”/>`
- Las relaciones entre clases se describen de la siguiente manera Suponiendo dos clases “Clase1” y “ClaseObjetivo”:

```
<Clase1 xmi:id=”Clase1_1”
ConexionConClaseObjetivo=”ClaseObjetivo_1”/>
<ClaseObjetivo xmi:id=”ClaseObjetivo_1” id=”instancia
CO1”/>
```

✓ **Utilización de XMI en herramientas case**

Al no tener una definición detallada en algunos aspectos del armado del XMI, como ser los parámetos obligatorios de la cabecera o utilizacion de namespaces estandares, siguiendo las especificaciones anteriores cada herramienta case genera su propio documento y frecuentemente no es posible el intercambio de modelos entre las distintas herramientas, 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.

Para el armado del XMI se toma el namespace que tiene asignado (xmlns:UML) en la cabecera del documento. Hay herramientas que asignan el namespace del estándar de UML, por ejemplo en el caso de “Enterprise Architect” (xmlns:uml=http://schema.omg.org/spec/UML/2.1). Sin embargo otras herramientas como lo es “ArgoUML” tienen sus propios namespaces.

Luego 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'>
```

Tomando un ejemplo de XMI que se desarrolla en la especificación de UML [OMG05a] la forma es similar a la que utiliza ArgoUML, utilizando “UML:Class” por ejemplo

```
<UML:Model xmi.id='S.1' name='Employment Model' visibility-'public' isSpecification-'false' isRoot-'false' isLeaf-'false' isAbstract-false'>
```

Aunque al tratar de importarlo en ambas herramientas, ninguna llegó a importarse. Dando en cada herramienta un error distinto.

En la documentación de la OMG [OMG05b] 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.

Otro inconveniente que surgió fue que al tomar ejemplos que se desarrollan dentro de los documentos de OMG, no se podían importar 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 importar. En otros casos informaban que no se podía importar ya que el XMI no estaba formateado correctamente sin detallar a qué se refería exactamente.

El formato XMI en la versión 2.1, que si bien el estándar 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.

3.3.4.1. Analizar los versionados de XMI.

La tabla 10 muestra una comparación de los versionados XMI a los que exporta cada uno de los Software.

Tabla 10 .Análisis de los versionados de XMI.

Herramienta	Versión	Fecha de publicación	XMI
Rational Software Architect	8.5	18/11/2011	2.1
Enterprise Architect	6.5.801	13/09/2006	1.1 ; 2.1
Enterprise Architect	9.1	20/09/2011	No indica
StarUML	5.0	30/12/2005	1.1
Dia Diagram Editor	0.97.2	18/12/2011	No soportado
Modelio	2.2.0	15/06/2012	OMG UML 2.1.1 ; 2.2 ; 2.3 ; 2.4.1 y EMF UML 3.0.0
Visual Paradigm	10.0	13/03/2012	1.0 ; 1.2 ; 2.1

3.3.4.2. Seleccionar el versionado de trabajo.

Seleccionamos la versión XMI 2.1 que es la que tiene más soportes en las herramientas de Modelado, como se puede observar en la tabla 10 del punto anterior.

3.3.4.3. Exportar cada modelo a las distintas herramientas en un XMI.

En esta sección se muestra el proceso realizado para exportar un modelo desde las distintas herramientas.

3.3.4.3.1. Enterprise Architect 6.5.801

✓ **Capturas de pantalla del modelo:**

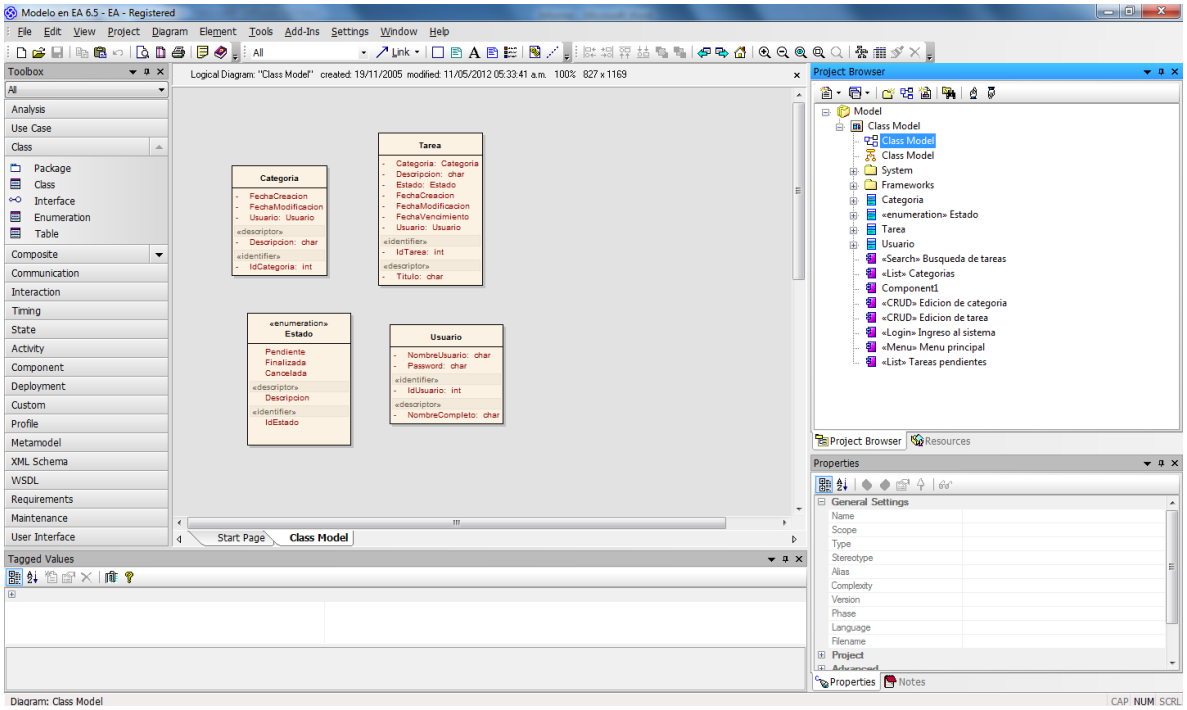


Figura 59. Pantalla 1 del modelo

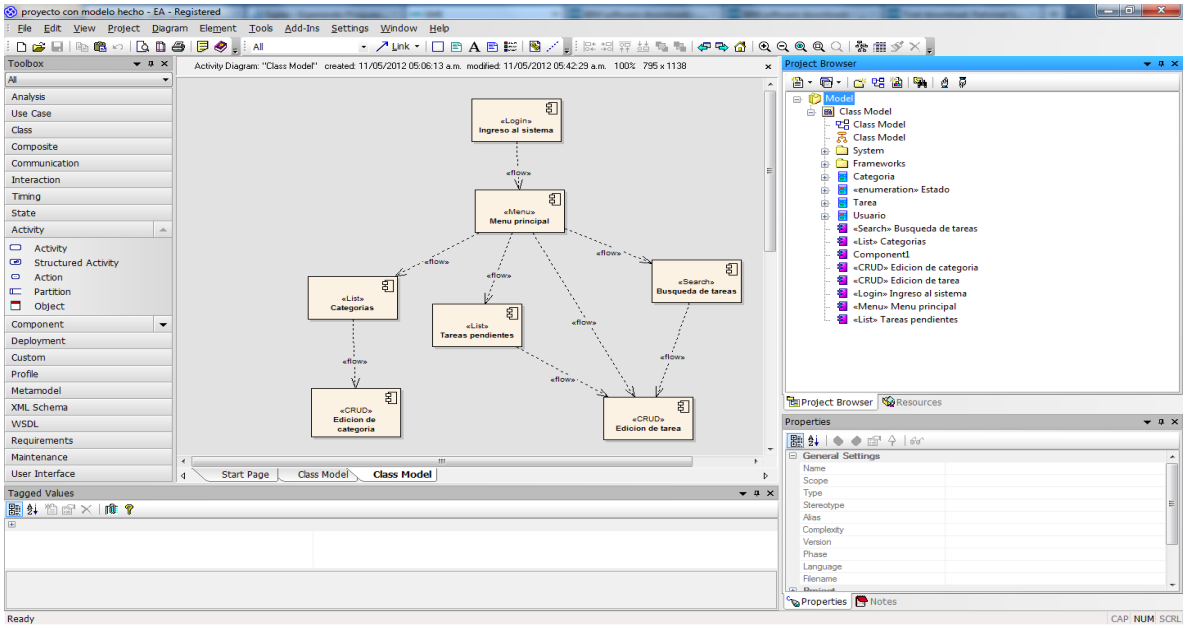


Figura 60. Pantalla 2 del Modelo.

✓ **Exportar:**

Click derecho sobre el modelo->Export model to XMI:

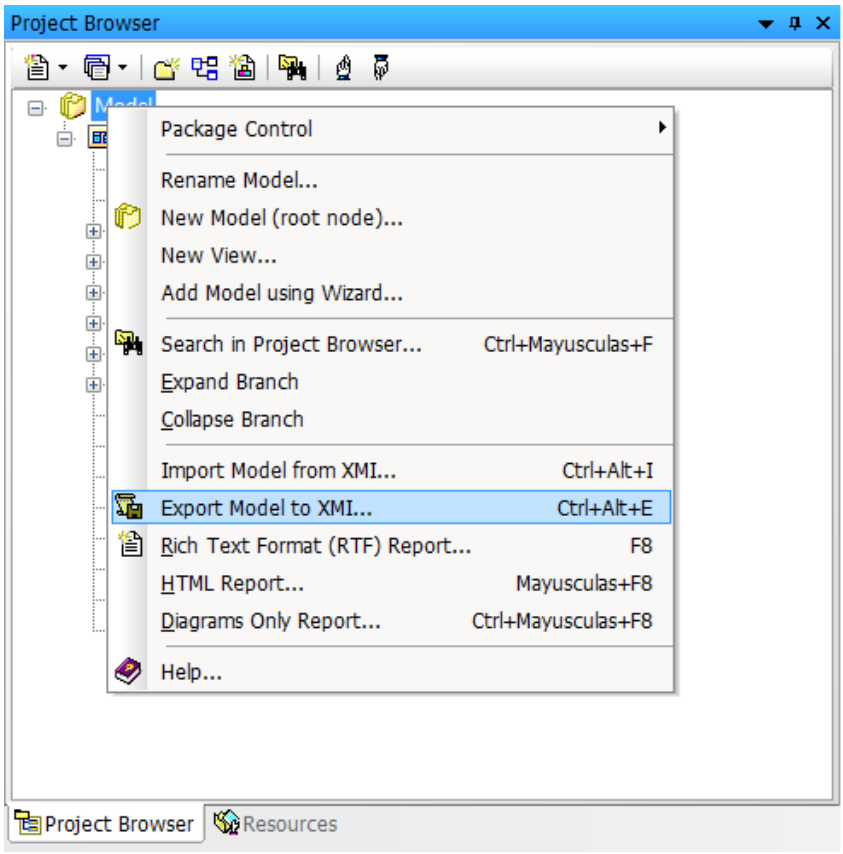


Figura 61. Pantalla 3 del Modelo.

Seleccionamos luego la versión de XMI a utilizar (1.1 o 2.1) :

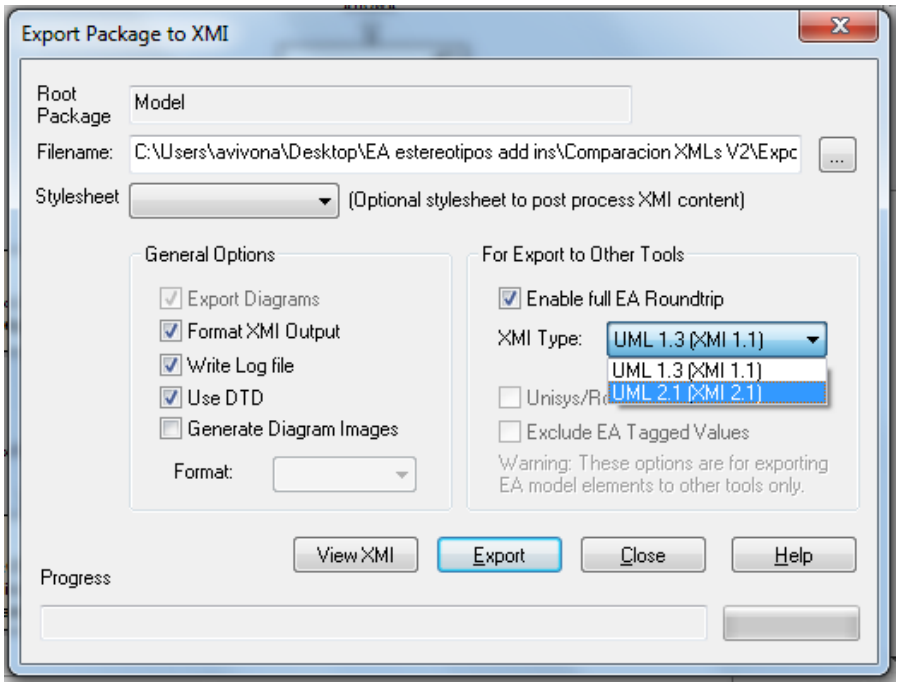


Figura 62. Pantalla 4 del Modelo

3.3.4.3.2. StarUML 5.0

✓ **Capturas de pantalla del modelo**

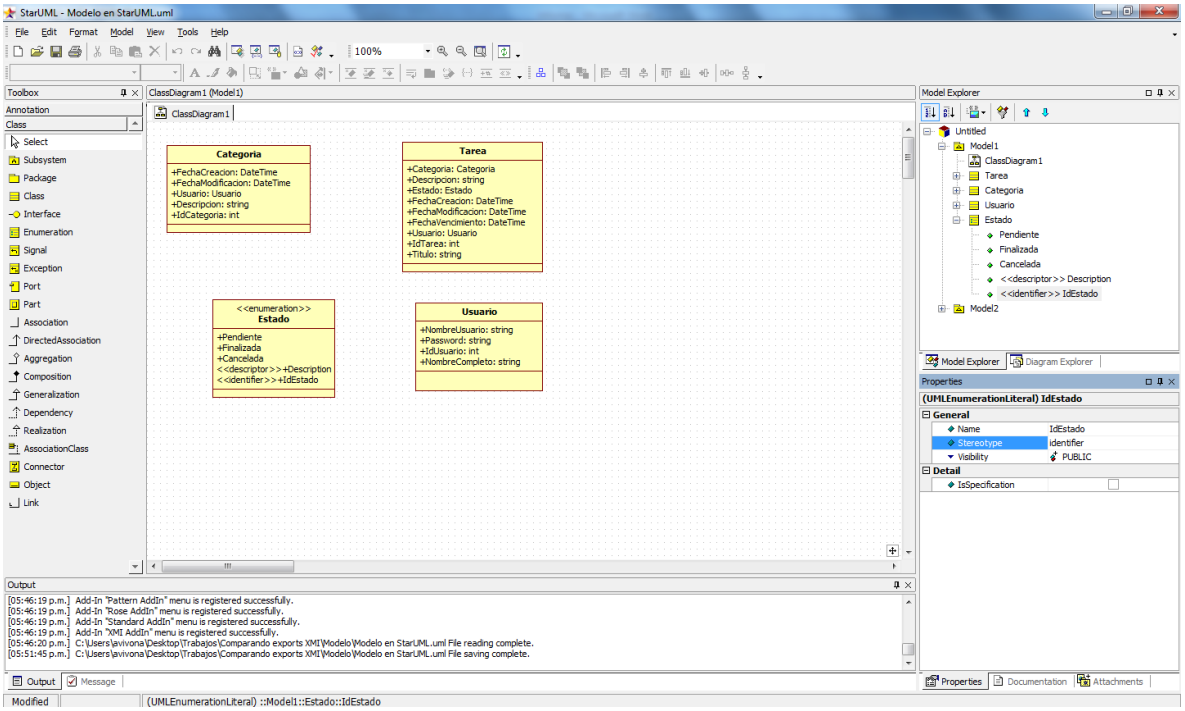


Figura 63. Pantalla1 del modelo Star UML 5.0

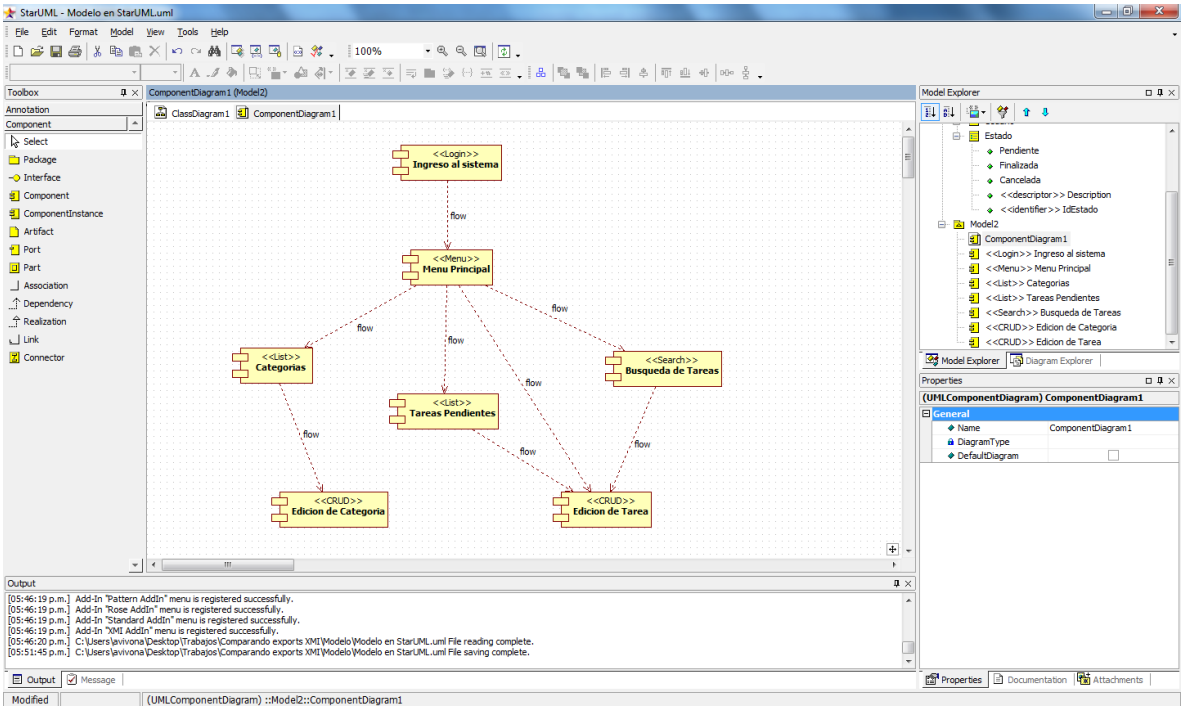


Figura 64. Pantalla 2 del modelo Star UML 5.0

✓ **Exportar**

Seleccionamos File->Export->XMI …

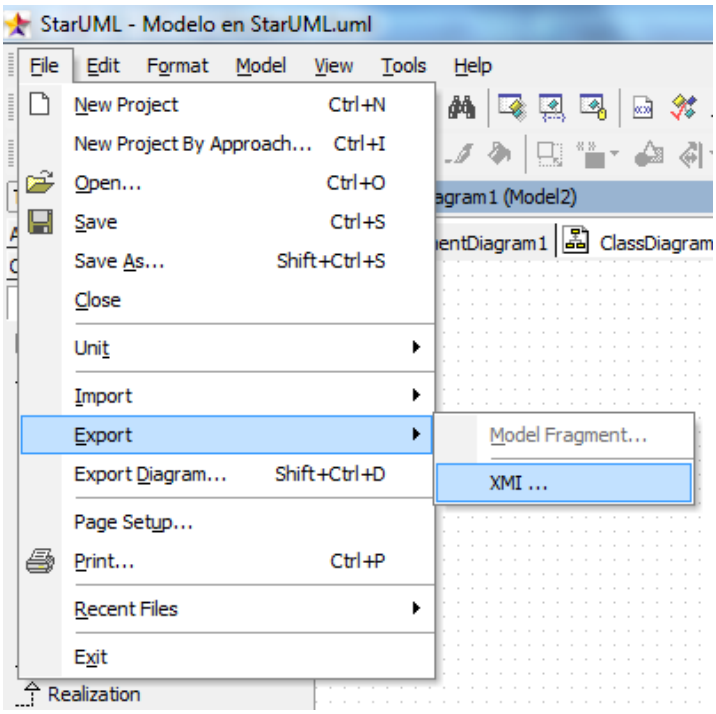


Figura 65. Exportar

Elegimos versión de XMI y directorio donde exportar

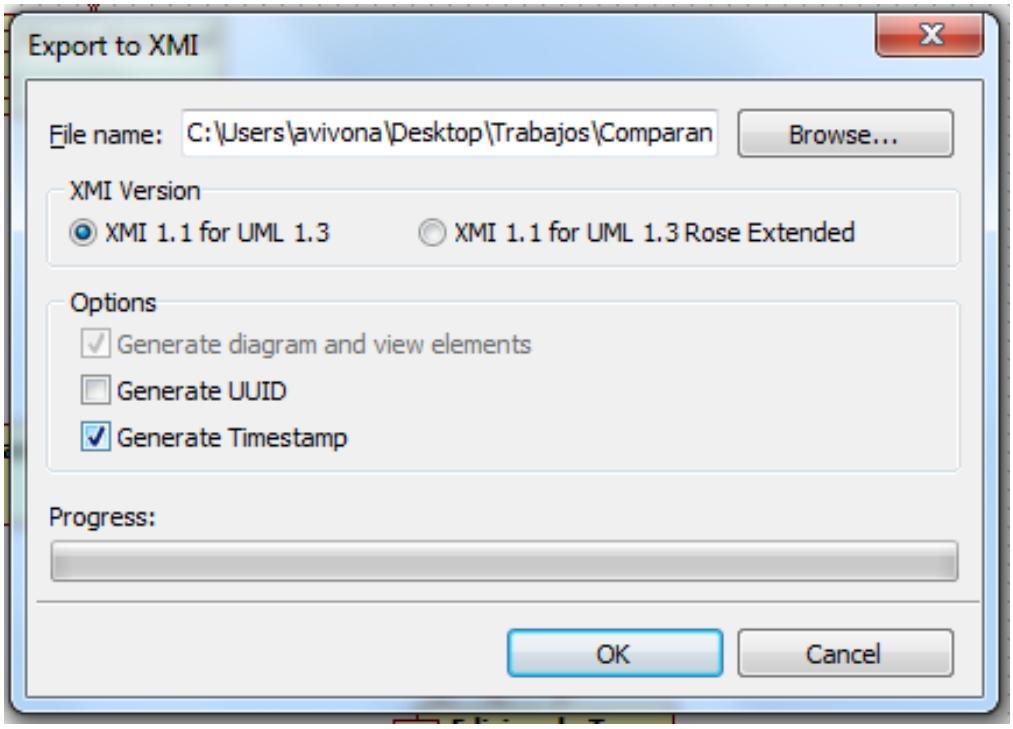


Figura 66. Exportar 2

3.3.4.3.3. Dia 0.97.2

✓ **Capturas de pantalla del modelo:**

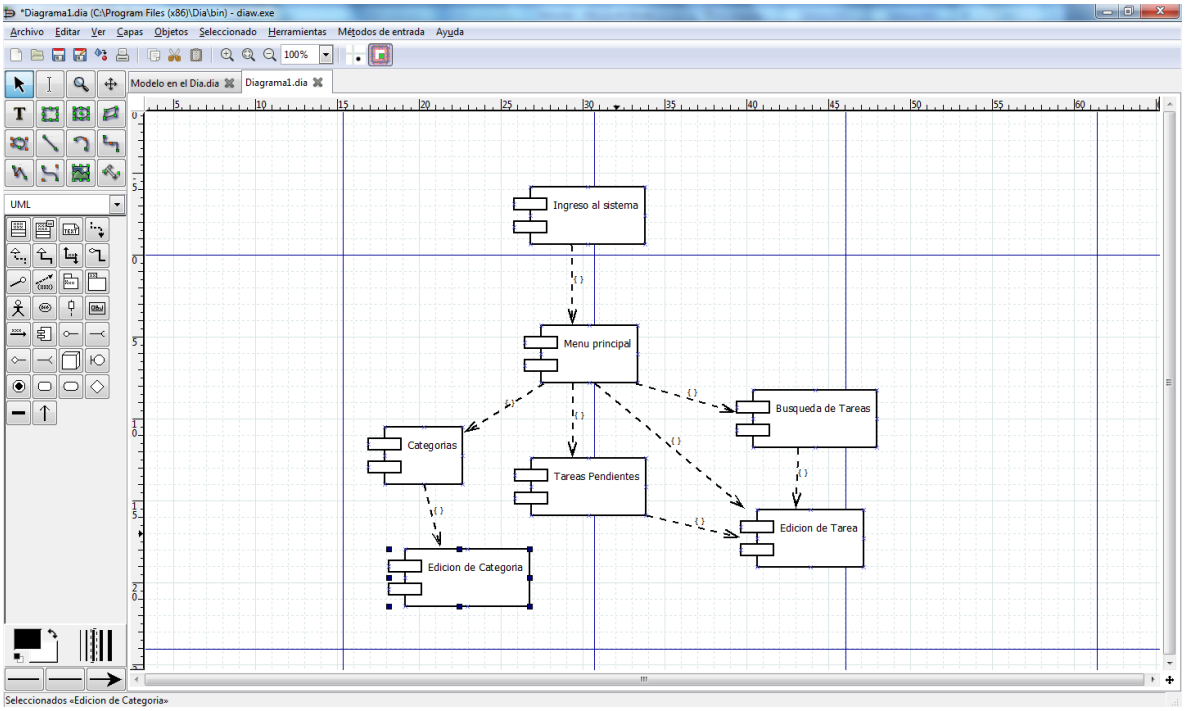


Figura 67. Captura 1. Dia

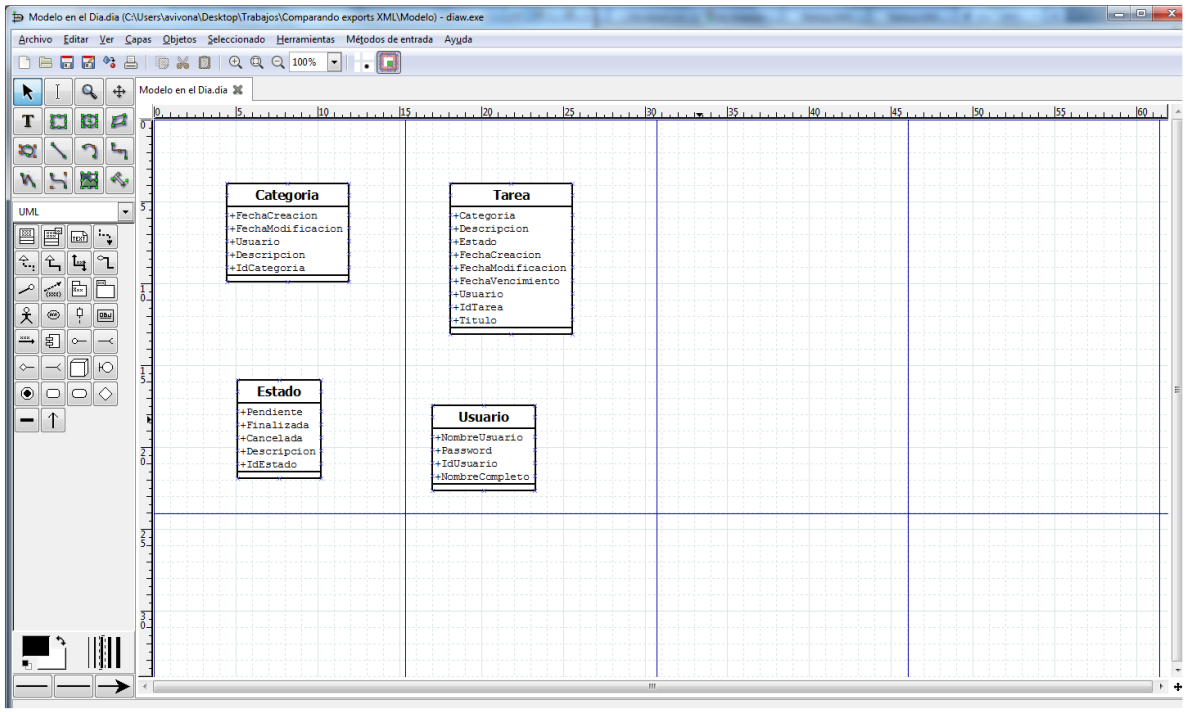


Figura 68. Captura 2. Dia

✓ **Exportar**

No se encuentra como exportar archivos desde la herramienta a XMI

Extraido de la ayuda:

“ Dia supports exporting to many other types of file, such as:

- *Computer Graphics Metafile (.cgm)*
- *Dia Native Diagram (.dia)*
- *Dia Shape File (.shape)*
- *AutoCad Drawing eXchange Format (.dxf)*
- *HP Graphics Language (.plt, .hpgl)*
- *Encapsulated Postscript (.eps, .epsi)*
- *Portable Network Graphics (.png)*
- *Scalable Vector Graphics (.svg)*
- *Scalable Vector Graphics gzip compressed (.svgz)*
- *TeX Metapost macros (.mp)*
- *TeX PSTricks macros (.tex)*
- *WordPerfect Graphics (.wpg)*
- *XFig format (.fig)*
- *XSLT (eXtensible Stylesheet Language Transformation) (.code) “*

3.3.4.3.4. Modelio 2.2.0

✓ **Capturas de pantalla del modelo:**

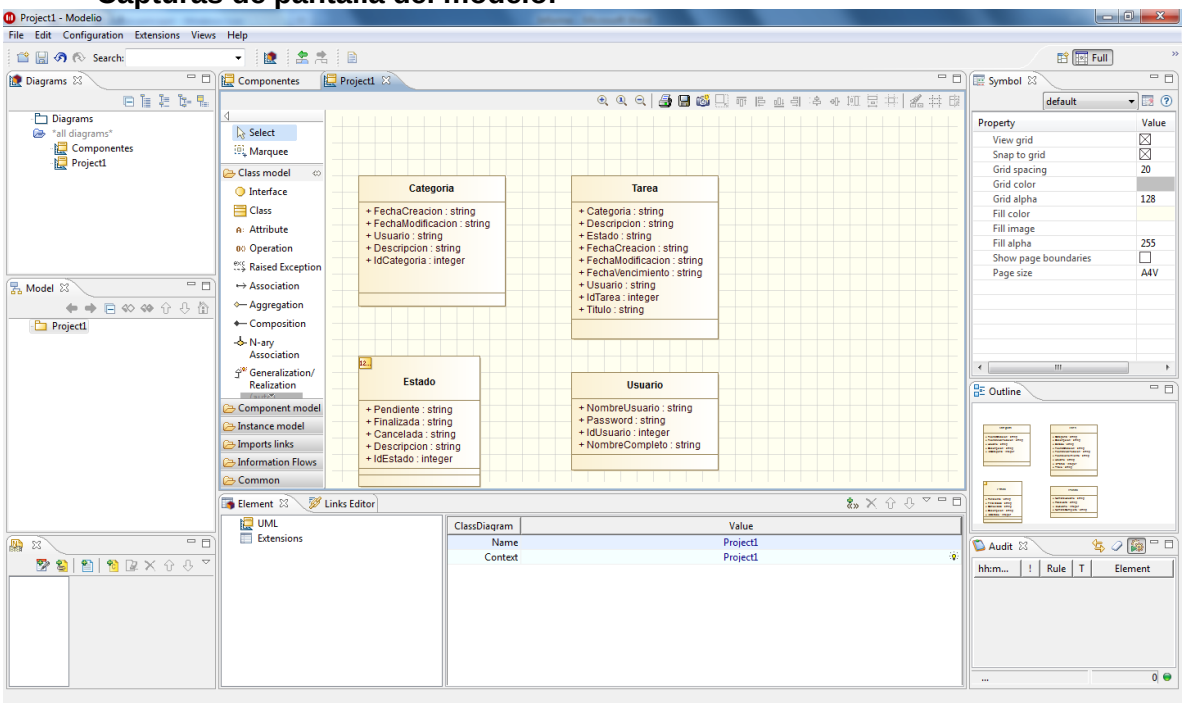


Figura 69. Captura 1. Modelio

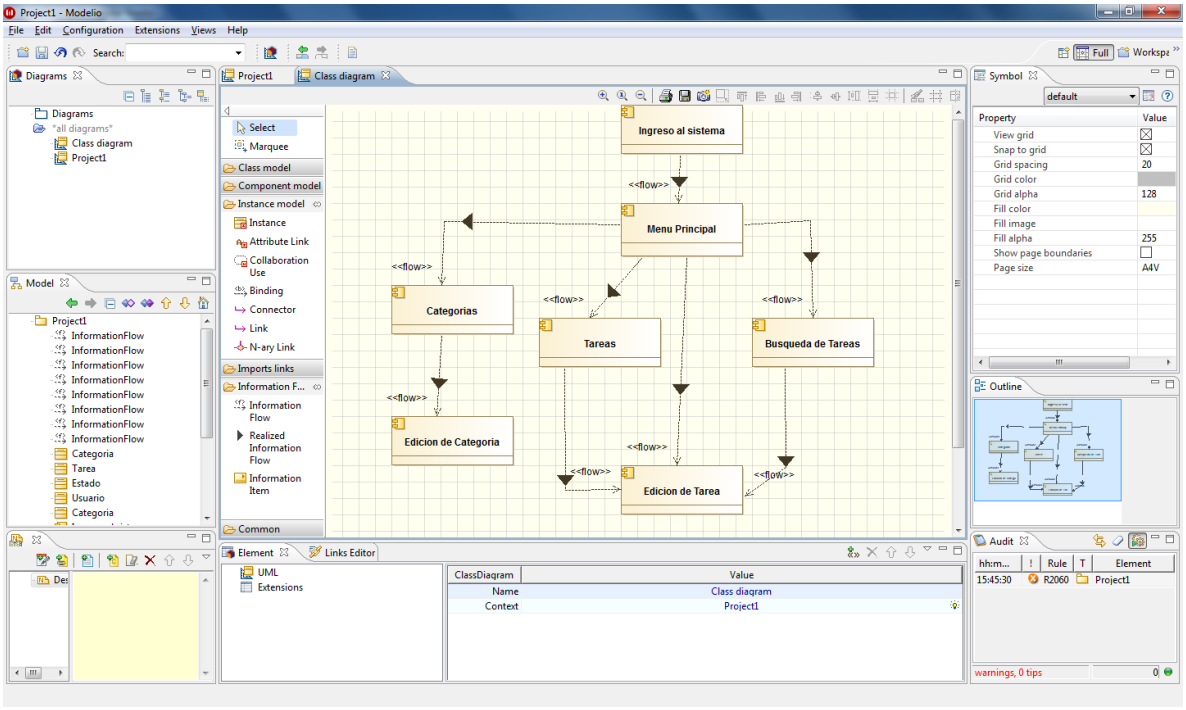


Figura 70. Captura 2. Modelio

- ✓ **Exportar:**

Sobre el proyecto seleccionamos: XMI->Export XMI

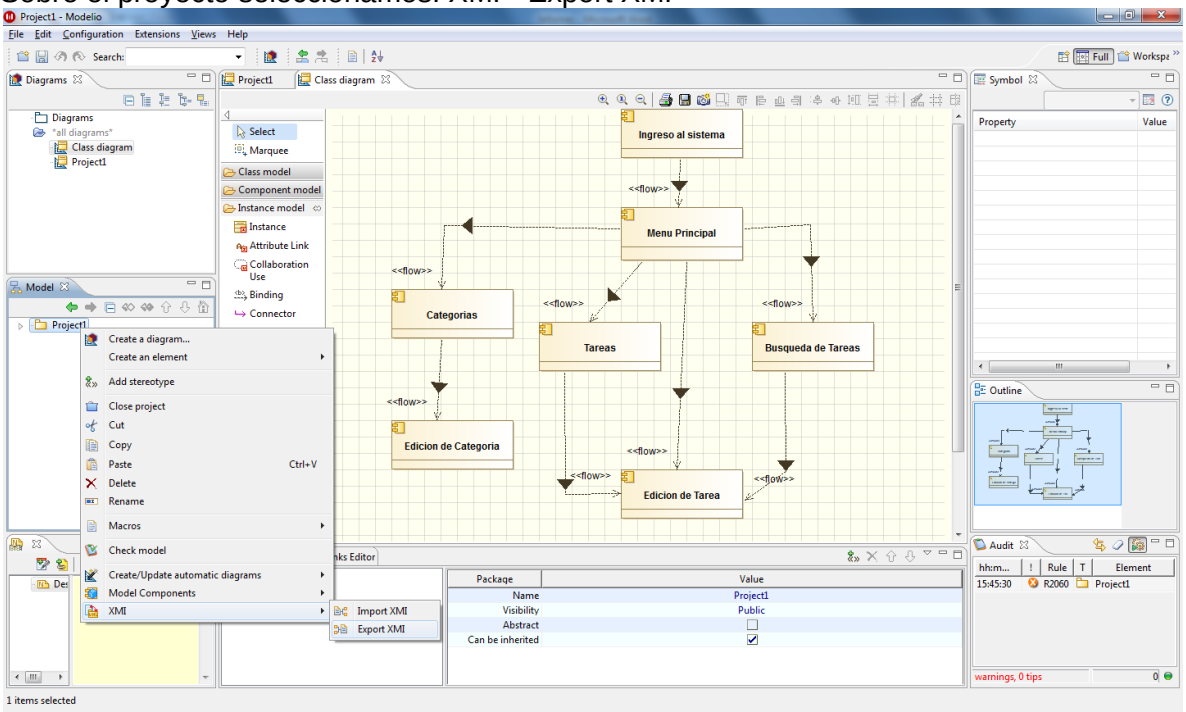


Figura 71. Exportar XMI.

Elegimos versión de XMI y directorio destino

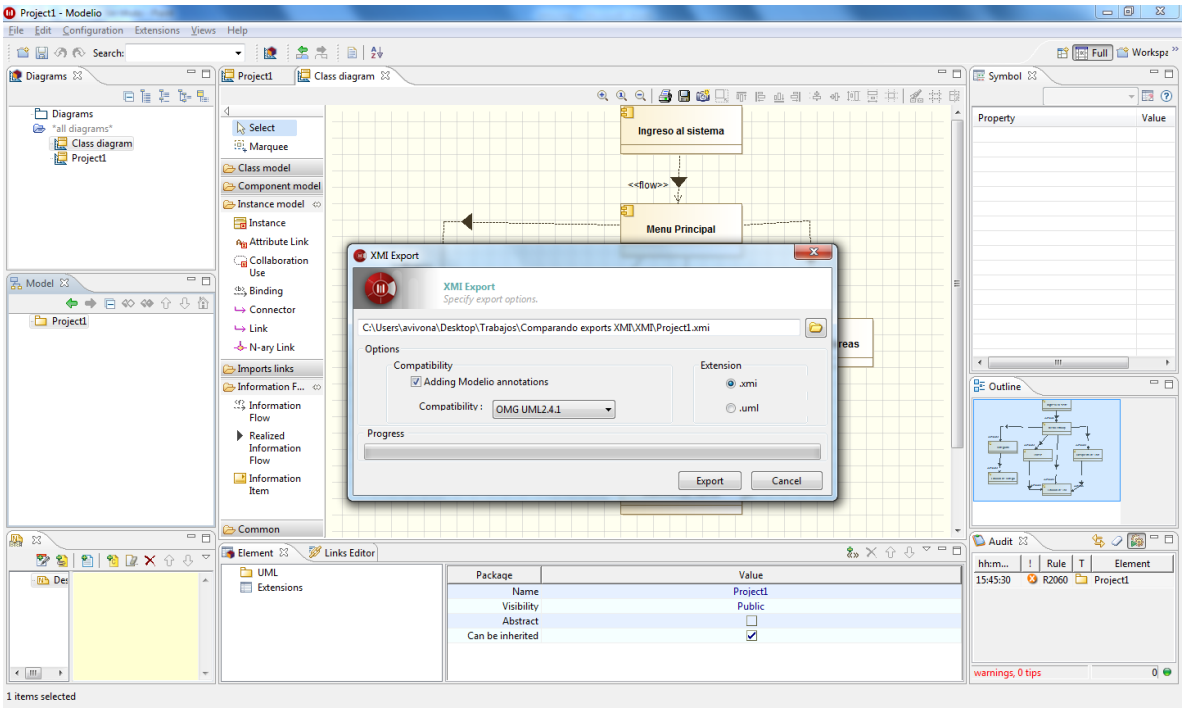


Figura 72. Elegir directorio destino en la exportación

3.3.4.3.5. Visual Paradigm 10.0

- ✓ **Capturas de pantalla del modelo:**

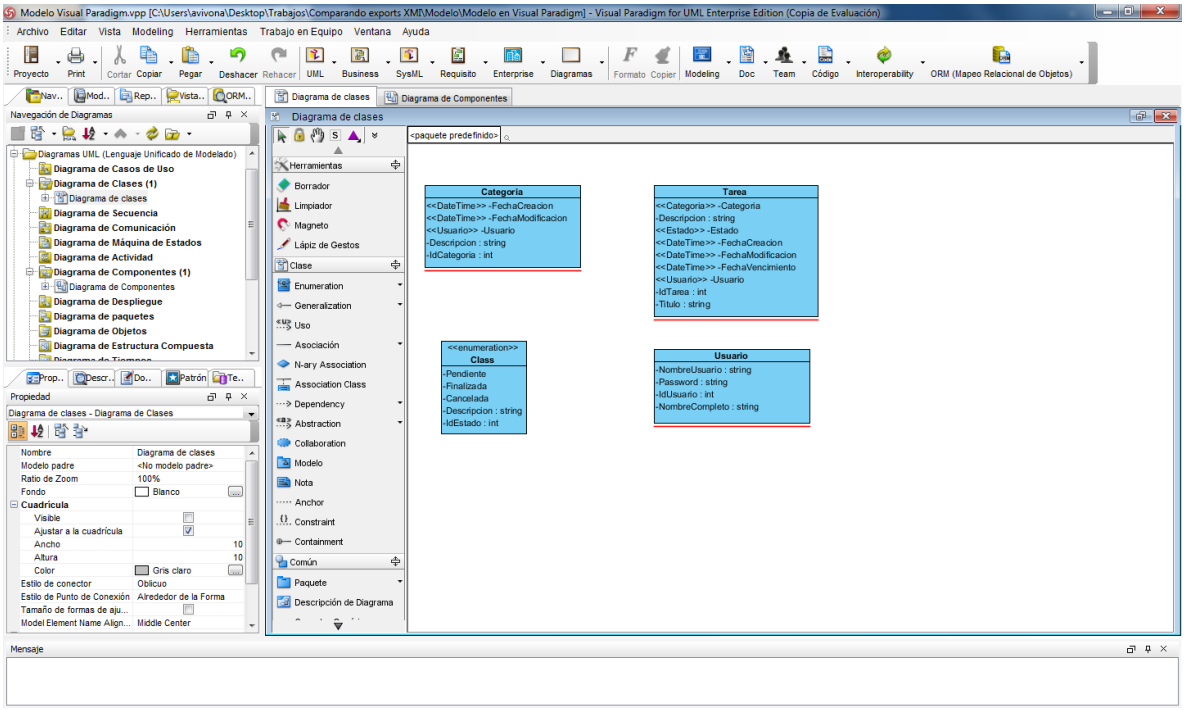


Figura 73. Captura 1. Visual Paradigm 10.0

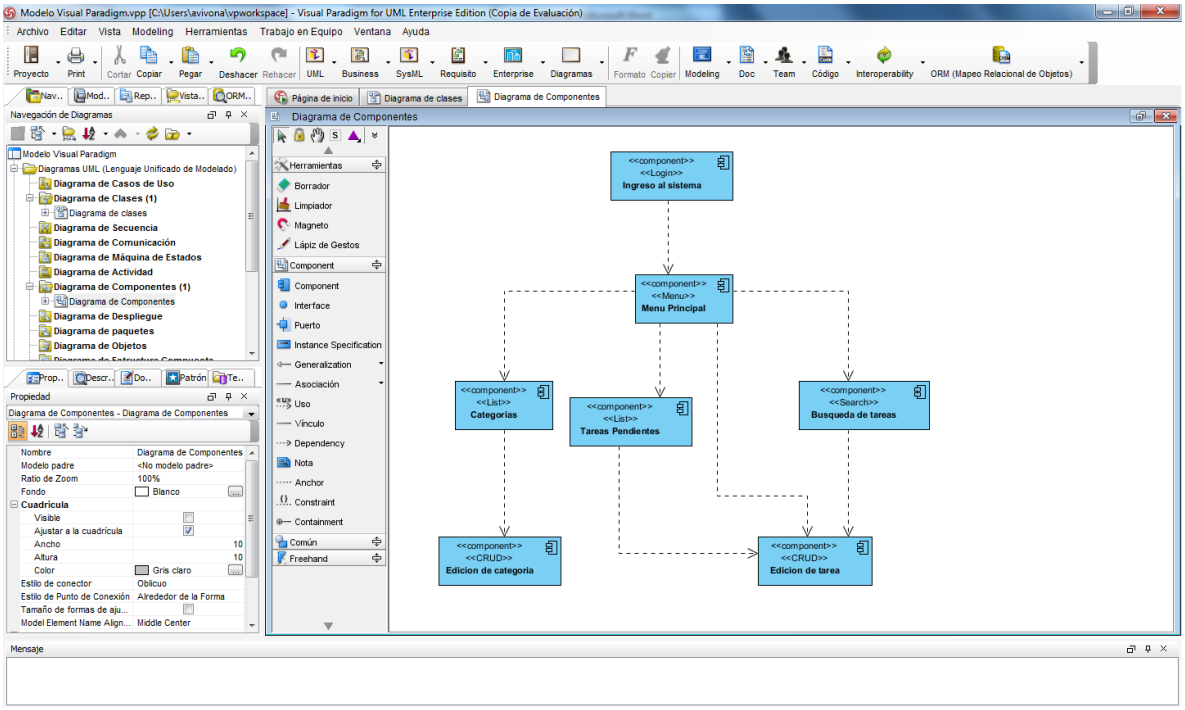


Figura 74. Captura 2. Visual Paradigm 10.0

- ✓ **Exportar:**

Seleccionamos: Interoperability->Exportar->XMI

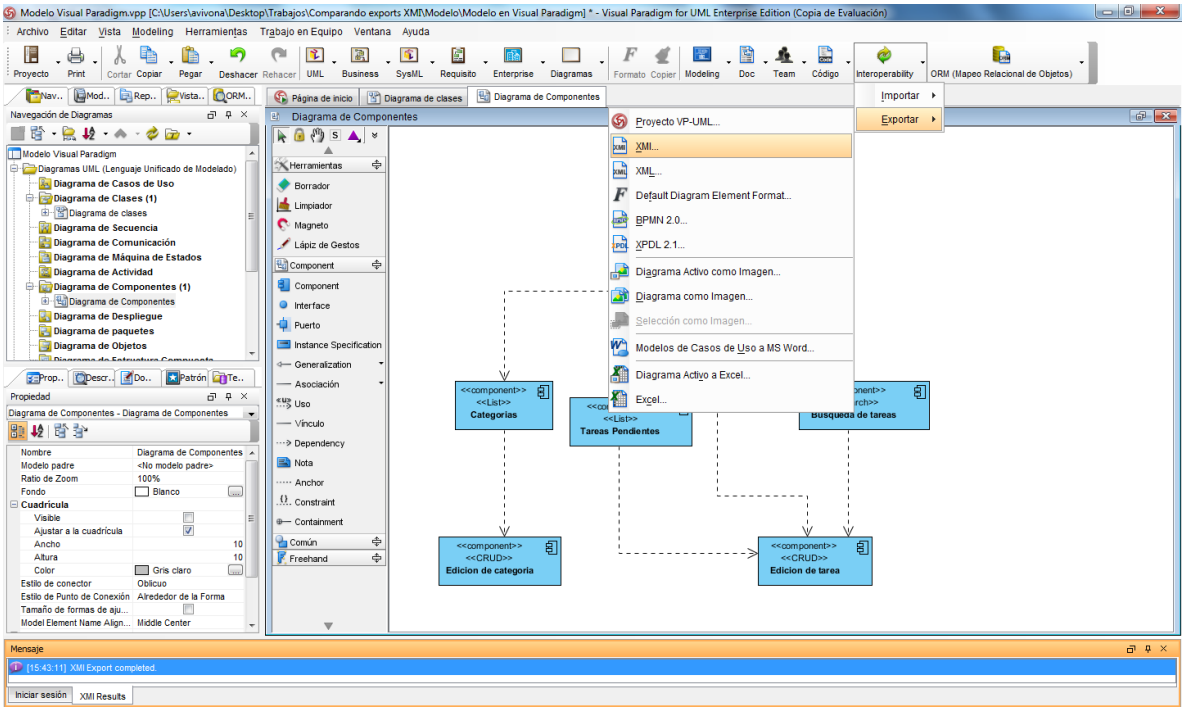


Figura 75. Exportar en Visual Paradigm 10.0

Seleccionamos version de XMI y directorio destino

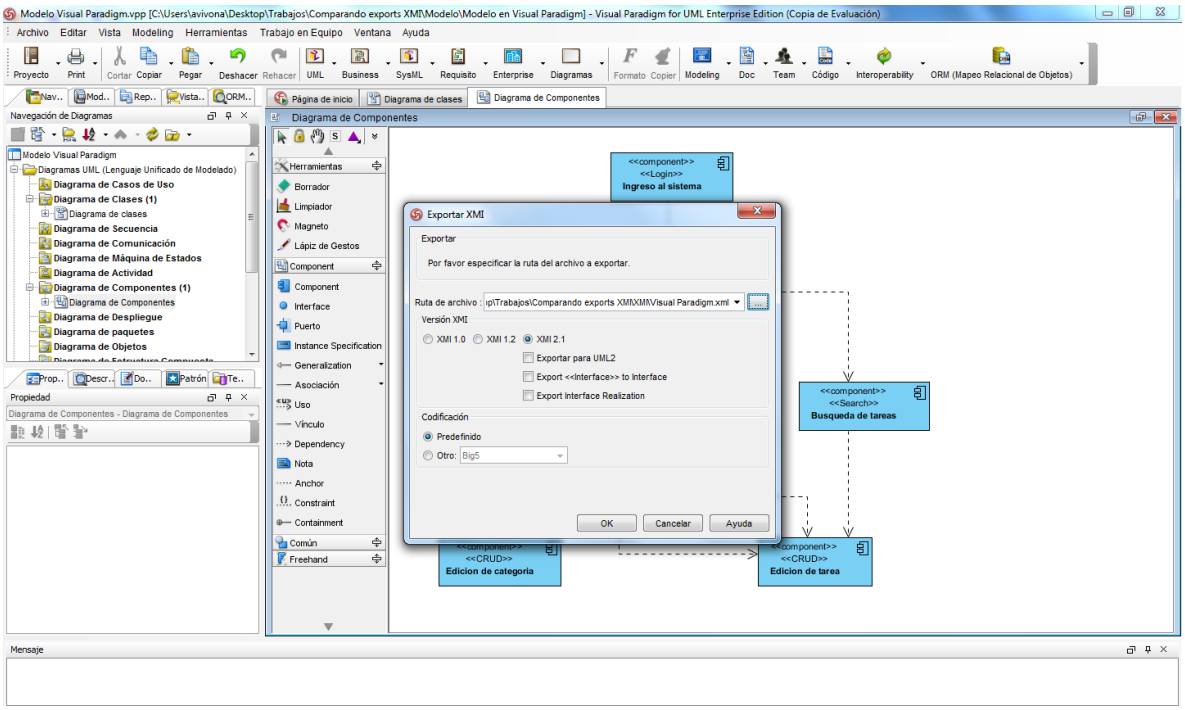


Figura 76. Elegir directorio en la xportación en Visual Paradigm 10.0

3.3.4.3.6. Rational Software Architect 8.5

✓ **Capturas de pantalla del modelo**

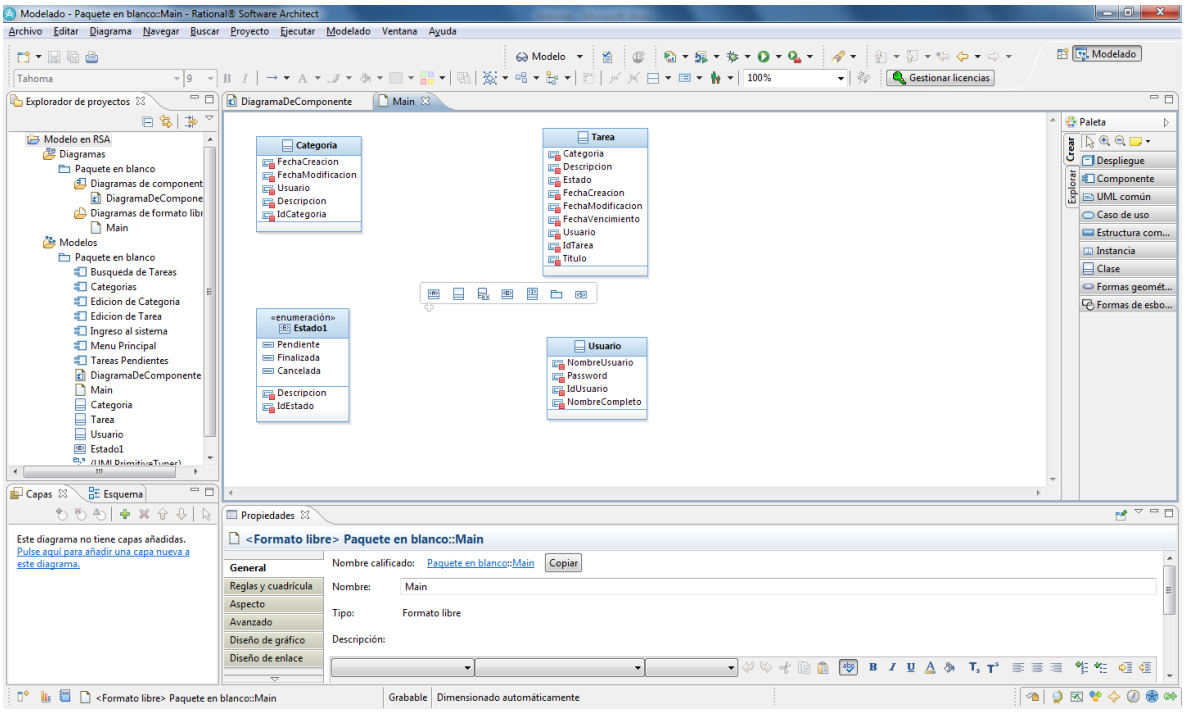


Figura 77. Captura de Pantalla 1 en Rational software Architect 8.5

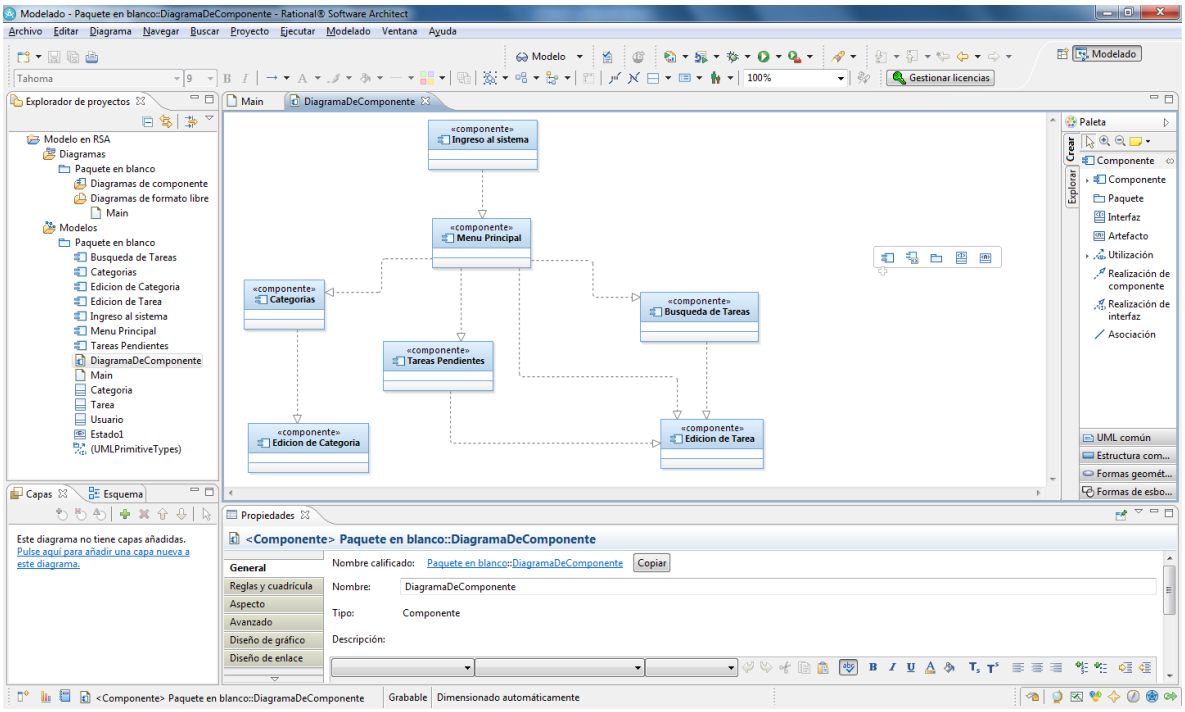


Figura 78. Captura de Pantalla 2 en Rational software Architect 8.5

- ✓ **Exportar:**

Haciendo click derecho sobre el proyecto seleccionamos “Exportar…”

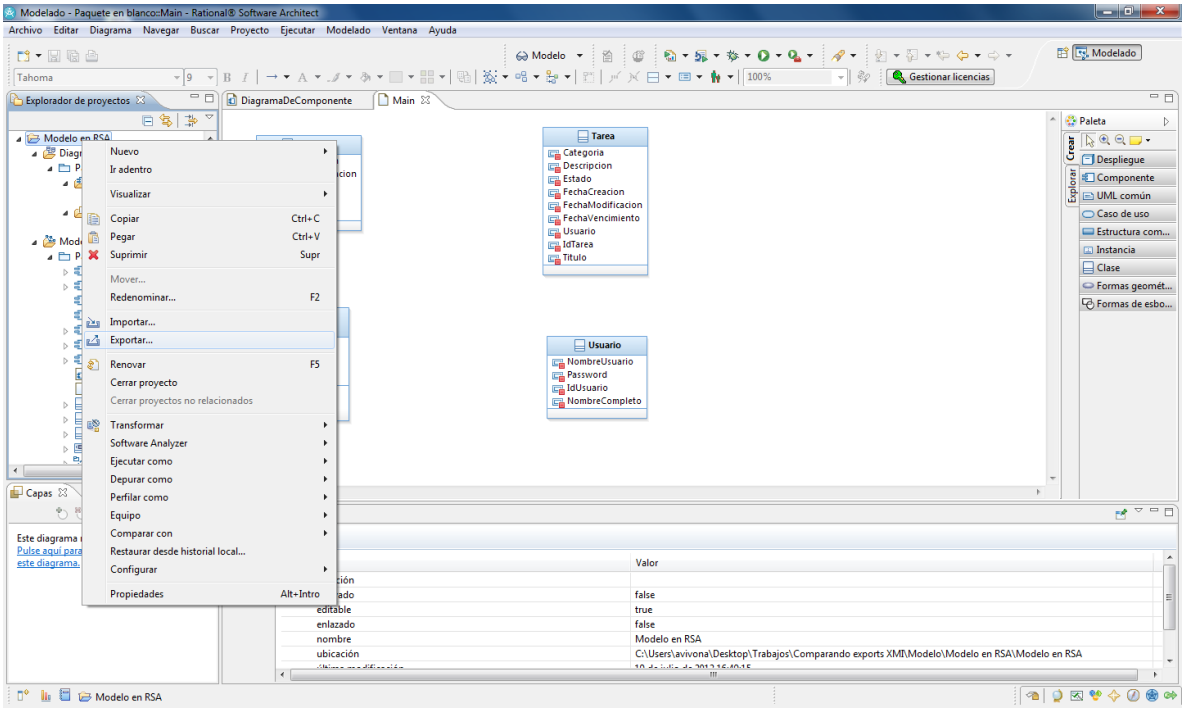


Figura 79. Pantalla 1: Exportar en Rational software Architect 8.5

En la ventana emergente nos dirigimos a Modelado->Modelo de intercambio de XMI de UML 2.2 y presionamos siguiente

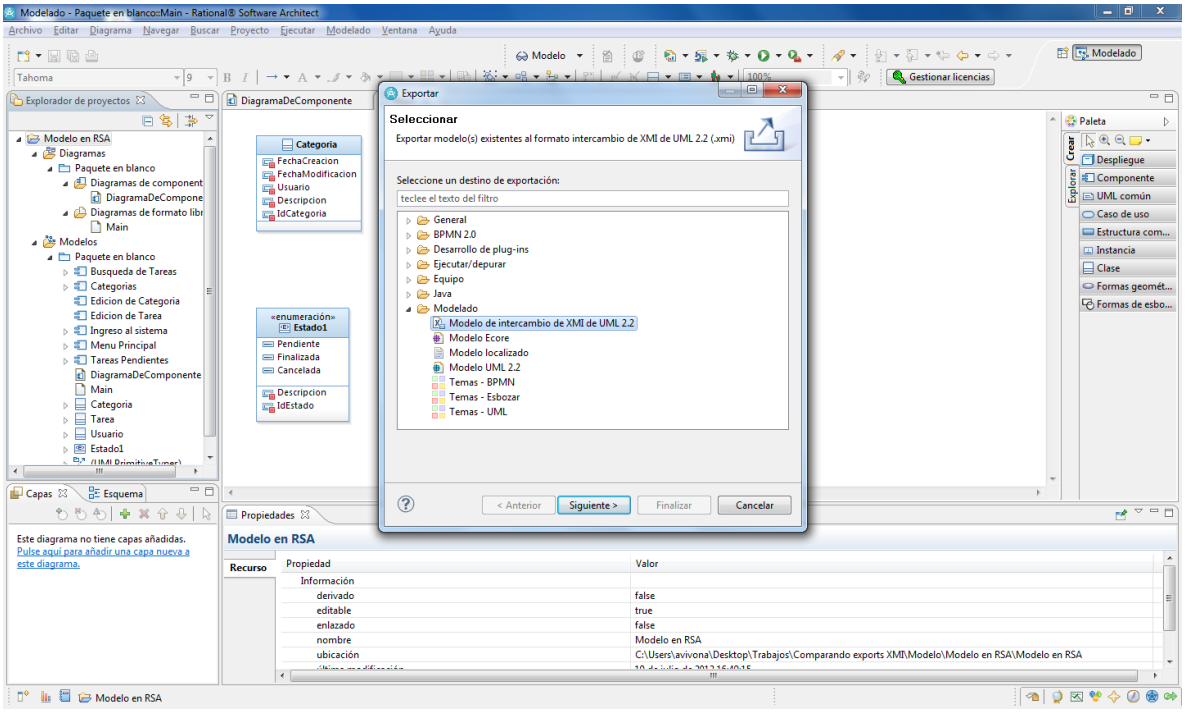


Figura 80. Pantalla 2 Exportar en Rational software Architect 8.5

Presionamos “examiner” en modelo

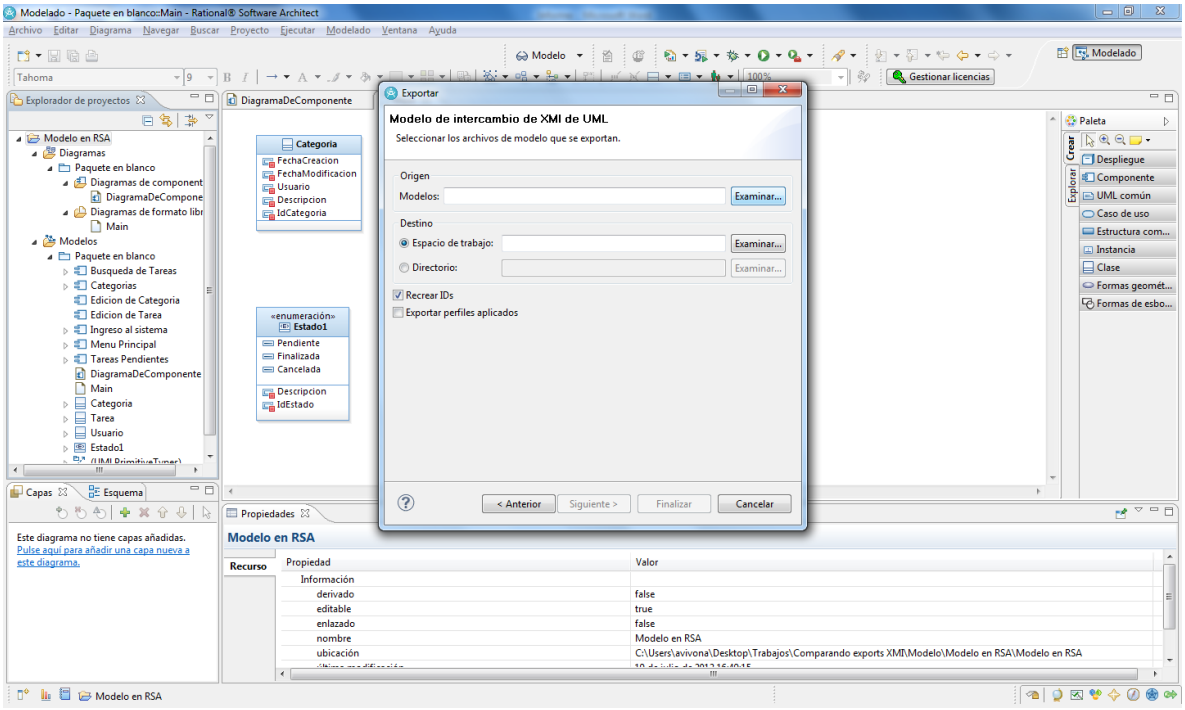


Figura 81. Pantalla 3 Exportar en Rational software Architect 8.5

Y elegimos el modelo a exportar, luego confirmamos

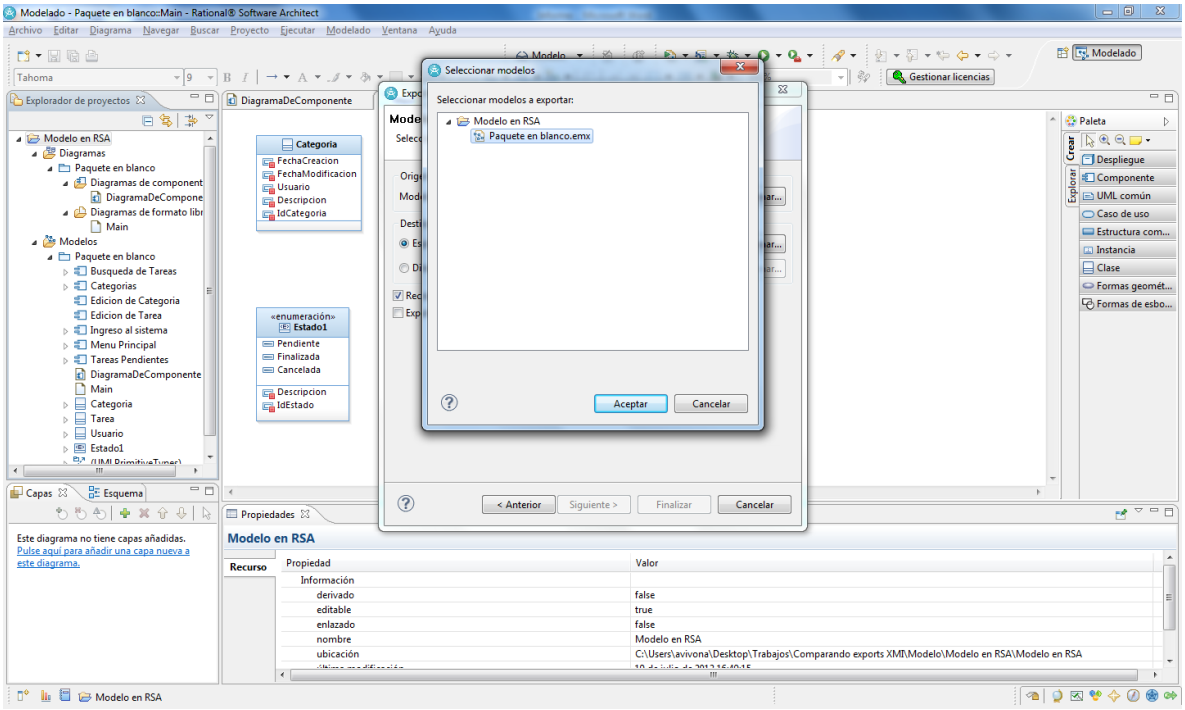


Figura 82. Pantalla 4 Exportar en Rational software Architect 8.5

Seleccionamos en “destino” el directorio destino del XMI a generar y presionamos aceptar

3.3.4.4. Analizar la correspondencia (mapeo). Elementos del Diagrama -> Elementos del XMI.

Tabla 11. XMI 1.2 (Argo UML) - XMI 1.1 (Enterprise Architect)

	XMI 1.2 (Argo UML)	XMI 1.1 (Enterprise Architect)
Estereotipos	El manejo de estereotipos se caracteriza porque estos se “definen” (es decir, su código se escribe) de forma independiente a los códigos que describen a los atributos/clases que los emplean.	A la hora de definir estereotipos, el código XMI de los mismos se caracteriza porque se define junto a los elementos (en los diagramas realizados, se hace con los atributos) que los emplean.
Tipos de datos	Su código, el cual indica su “xmi.id”, nombre, si es abstracto, etc; se define de forma separada al código que describe a los	En el caso del uso de valores etiquetados en el Enterprise Architect , su código XMI correspondiente se caracteriza

	elementos/atributos/clases que lo contienen.	debido a que la definición del mismo se “separa” de su uso (se define al final del código XMI, pero se hace mención de su utilización en los correspondientes atributos)
Valores de Enumeración		El código XMI de un valor de enumeración se caracteriza porque, a cada valor de enumeración, se lo define como un atributo, el cual tiene un estereotipo “enum” (correspondiente al nombre que se le ha dado al estereotipo) que indica que es un valor de enumeración.
Valores Etiquetados	El código XMI generado, se caracteriza porque describe las características de determinado valor etiquetado (por ejemplo: su “xmi.id”). Por otro lado, el código se caracteriza porque se define de forma separada al código respectivo al elemento (clase, atributo) que lo emplea.	La definición de un valor etiquetado se caracteriza porque el código correspondiente a la misma se encuentra definido fuera del código XMI que corresponde a los atributos que emplean dicho valor etiquetado.

Tabla 12. XMI 2.1 (Enterprise Architect) - XMI 2.1 (Magic Draw).

	XMI 2.1 (Enterprise Architect)	XMI 2.1 (Magic Draw)
Estereotipos	Los códigos XMI de los estereotipos se caracterizan porque los mismos se definen individualmente (o sea, fuera del código correspondiente a los atributos que lo emplean). Por otra parte, se caracterizan porque a la hora de utilizarlos se lo hace de forma separada a su definición (es decir, se indica en el código correspondiente al atributo que lo emplea el nombre de dicho estereotipo).	El código XMI correspondiente a un Estereotipo se caracteriza porque se define "Fuera" del código XMI que corresponde al atributo que lo emplea.
Tipos de datos	Los tipos de datos se definen en el código XMI que corresponde a los atributos que lo emplean.	En el caso de la definición de un tipo de dato, esta se caracteriza debido a que se define dentro del código XMI que corresponde a la Clase que la emplea.
Valores de Enumeración	A un valor de enumeración se lo define dentro del código XMI que corresponde al atributo que lo emplea.	A la hora de definir el código XMI que corresponde a un Valor de enumeración, este se debe de definir dentro del código de la clase que lo emplea.
Valores Etiquetados	El código XMI que corresponde a un valor etiquetado se define dentro del código que corresponde al atributo que emplea dicho valor etiquetado.	A los códigos de XMI de los Valores Etiquetados se los define dentro del código XMI que corresponde a la clase o atributo que emplea dicho valor etiquetado.

Xmi 1.2 (Argo Uml)

A. Estereotipo

* ***Estereotipo Descriptor***

Código:

<UML:Stereotype xmi.id = '-64--88-0-10--55becee9:13f5226b73a: 8000:0000000000000AB1'

name = **'Descriptor'** isSpecification = 'false' isRoot = 'false' isLeaf = 'false'

isAbstract = 'false'>

<UML:Stereotype.baseClass>Operation</UML:Stereotype.baseClass>

</UML:Stereotype>

Descripción:

Al estereotipo **Descriptor** se lo define de forma “individual”, es decir, no se define dentro de otros elementos (ya sean clases, atributos, etc).

* ***Estereotipo Identifier***

Código:

<UML:Stereotype xmi.id = '-64--88-0-10--55becee9:13f5226b73a: 8000:0000000000000AB2'

name = **'Identifier'** isSpecification = 'false' isRoot = 'false' isLeaf = 'false'

isAbstract = 'false'>

<UML:Stereotype.baseClass>Operation</UML:Stereotype.baseClass>

</UML:Stereotype>

Descripción:

Al estereotipo **Identifier** se lo define de forma “individual”, es decir, no se define dentro de otros elementos (ya sean clases, atributos, etc).

B. Tipo de Dato

* ***Tipo de dato cbhdmString***

Código:

<UML:DataType isAbstract="**false**" isLeaf="**false**" isRoot="**false**" isSpecification="**false**" name="**cbhdmString**" xmi.id="**-64--88-0-10--55becee9:13f5226b73a:-8000:0000000000000AA2**" visibility="**package**"/>

Descripción:

Al tipo de dato cbhdmString, a la hora de definirlo, se lo hace de forma “individual”, es decir, se lo hace “fuera” de los elementos descriptos en el código (ya sean clases, atributos, etc.) que lo emplean.

* ***Tipo de dato cbhdmint***

Código:

<UML:DataType isAbstract="**false**" isLeaf="**false**" isRoot="**false**" isSpecification="**false**" name="**cbhdmInt**" xmi.id="**-64--88-0-10--55becee9:13f5226b73a:-8000:0000000000000AA3**" visibility="**package**"/>

Descripción:

El tipo de dato cbhdmint se define individualmente, o sea, se lo hace de forma aparte respecto de los elementos que lo utilizan.

* ***Tipo de dato cbhdmDateTime***

Código:

<UML:DataType isAbstract="**false**" isLeaf="**false**" isRoot="**false**" isSpecification="**false**" name="**cbhdmDateTime**" xmi.id="**-64--88-0-10--55becee9:13f5226b73a:-8000:0000000000000AA4**" visibility="**package**"/>

Descripción:

De forma similar que con los tipos de datos cbhdmint y cbhdmstring, al tipo de dato cbhdmDateTime se lo define individualmente, o sea, de forma separada respecto a los elementos (atributos, operaciones, etc.) que lo utilizan.

C. Valores Etiquetados

* ***Valor etiquetado "NombreUsuario"***

Código:

<UML:TaggedValue xmi.id = '-64--88-0-10--725daaa5:13f765dfd3a:-8000:000000000000098C' isSpecification = 'false'>

<UML:TaggedValue.dataValue>NombreUsuario</UML:TaggedValue.dataValue>

<UML:TaggedValue.type>

<UML:TagDefinition xmi.idref = '-64--88-0-10--725daaa5:13f765dfd3a:-8000:0000000000000989'/>

</UML:TaggedValue.type>

</UML:TaggedValue>

Descripción:

Este valor etiquetado se define de forma independiente, o sea, no se define dentro del código que corresponde a la definición de aquellos elementos (atributos, clases, propiedades) que lo utilizan.

* ***Valor etiquetado "Clave"***

Código:

<UML:TaggedValue xmi.id = '-64--88-0-10--725daaa5:13f765dfd3a:-8000:0000000000000990'

isSpecification = 'false'>

<UML:TaggedValue.dataValue>Clave

</UML:TaggedValue.dataValue>

<UML:TaggedValue.type>

<UML:TagDefinition xmi.idref = '-64--88-0-10--725daaa5:13f765dfd3a:-8000:000000000000098D'/>

</UML:TaggedValue.type>

</UML:TaggedValue>

Descripción:

Este valor etiquetado se define de forma independiente, o sea, no se define dentro del código que corresponde a la definición de aquellos elementos (atributos, clases, propiedades) que lo utilizan.

* ***Valor etiquetado "Ingreso al Sistema"***

Código:

<UML:TaggedValue xmi.id = '-64--88-0-10--725daaa5:13f765dfd3a:-8000:0000000000000994'

isSpecification = 'false'>

<UML:TaggedValue.dataValue>Ingreso al Sistema</UML:TaggedValue.dataValue>

<UML:TaggedValue.type>

<UML:TagDefinition xmi.idref = '-64--88-0-10--725daaa5:13f765dfd3a:-8000:0000000000000991'/>

</UML:TaggedValue.type>

</UML:TaggedValue>

Descripción:

Este valor etiquetado se define de forma independiente, o sea, no se define dentro del código que corresponde a la definición de aquellos elementos (atributos, clases, propiedades) que lo utilizan.

3.3.4.4.1. Xmi 1.1 (Enterprise Architect)

A. Estereotipo

* ***Estereotipo Descriptor***

Código:

```
<UML:Classifier.feature>
        <UML:Attribute name="Descripcion" changeable="none"
visibility="private" ownerScope="instance" targetScope="instance">
   <UML:Attribute.initialValue>
        <UML:Expression/>
      </UML:Attribute.initialValue>
      <UML:StructuralFeature.type>
            <UML:Classifier xmi.idref="eaxmiid0"/>
        </UML:StructuralFeature.type>
        <UML:ModelElement.stereotype>
              <UML:Stereotype name="Descriptor"/>
        </UML:ModelElement.stereotype>
        <UML:ModelElement.taggedValue>
            <UML:TaggedValue tag="type" value="cbhdmString"/>
                        /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
        </UML:ModelElement.taggedValue>
        </UML:Attribute>
```

Descripción:

Al estereotipo **Descriptor** no se lo define de forma “individual”, ya que su definición se incluye dentro de los atributos (en el caso descripto anteriormente, **“Descripción”**) que lo emplean.

* ***Estereotipo Identifier***

Código:

```
<UML:Attribute name="IdEstado" changeable="none" visibility="private"
ownerScope="instance" targetScope="instance">
                        <UML:Attribute.initialValue>
```

<UML:Expression/>
</UML:Attribute.initialValue>
<UML:StructuralFeature.type>
<UML:Classifier xmi.idref="eaxmiid1"/>
</UML:StructuralFeature.type>
<UML:ModelElement.stereotype>
<UML:Stereotype name="Identifier"/>
</UML:ModelElement.stereotype>
<UML:ModelElement.taggedValue>
<UML:TaggedValue tag="type" value="cbhdmInt"/>
<UML:TaggedValue tag="stereotype" value="Identifier"/>

/\
<UML:TaggedValue tag="$ea_xref_property" value="$XREFPROP=$XID={2DB02C25-A710-4246-9D57-875996835FED}$XID;$NAM=Stereotypes$NAM;$TYP=attribute property$TYP;$VIS=Public$VIS;$DES=@STEREO;Name=Identifier;GUID={690C6F40-B808-498f-B9D2-85955051C335};@ENDSTEREO;$DES;$CLT={47EB8FC8-8CEA-43e5-9D0A-1EDCEE681905}$CLT;$SUP=<none>$SUP;$ENDXREF;"/>
</UML:ModelElement.taggedValue>
</UML:Attribute>

Descripción:

Al estereotipo **Identifier** no se lo define de forma “individual”, ya que su definición se incluye dentro de los atributos (en el caso descripto anteriormente, **“IdEstado”**) que lo emplean.

B. Tipos de datos

* ***Tipo de dato cbhdmstring***

Definición:

<UML:DataType xmi.id="**eaxmiid0**" name="**cbhdmString**" isAbstract="**false**" isLeaf="**false**" isRoot="**false**" visibility="**private**"/>

Uso:

<UML:Classifier.feature>
<UML:Attribute name="Descripcion" changeable="none" visibility="private" ownerScope="instance" targetScope="instance">

/\

<UML:ModelElement.taggedValue>
<UML:TaggedValue tag="type" value="cbhdmString"/>
<UML:TaggedValue tag="derived" value="0"/>

/\

<UML:TaggedValue tag="$ea_xref_property" value="$XREFPROP=$XID={06ED7722-9398-4d33-9EB9-C839F7CA5235}$XID;$NAM=Stereotypes$NAM;$TYP=attribute property$TYP;$VIS=Public$VIS;$DES=@STEREO;Name=Descriptor;GUID ={4B3236F1-0B95-406f-81E6-

D2375E26DC84};@ENDSTEREO;$DES;$CLT={96E645DC-C36D-4639- A4A9-9A128BC58DD7}$CLT;$SUP=<none>$SUP;$ENDXREF;"/>

</UML:ModelElement.taggedValue>
</UML:Attribute>

Descripción:

Al tipo de dato cbhdmstring, para este caso, se lo define al final del código xmi, pero al utilizarlo, se lo hace dentro del código correspondiente al atributo que lo utiliza, en la parte de “Valores etiquetados” (“Tagged values”) [<UML:TaggedValue tag="type" value="cbhdmString"/>]

* ***Tipo de dato cbhdmint***

Definición:

<UML:DataType xmi.id="**eaxmiid1**" name="**cbhdmInt**" isAbstract="**false**" isLeaf="**false**" isRoot="**false**" visibility="**private**"/>

Uso:

<UML:Attribute name="IdEstado" changeable="none" visibility="private" ownerScope="instance" targetScope="instance">

<UML:Attribute.initialValue>

/\

<UML:ModelElement.taggedValue>

<UML:TaggedValue tag="type" value="cbhdmInt"/>

<UML:TaggedValue tag="derived" value="0"/>

/\

<UML:TaggedValue tag="$ea_xref_property" value="$XREFPROP=$XID={2DB02C25-A710-4246-9D57-875996835FED}$XID;$NAM=Stereotypes$NAM;$TYP=attribute property$TYP;$VIS=Public$VIS;$DES=@STEREO;Name=Identifier;GUID={690C6F40-B808-498f-B9D2-85955051C335};@ENDSTEREO;$DES;$CLT={47EB8FC8-8CEA-43e5-9D0A-1EDCEE681905}$CLT;$SUP=<none>$SUP;$ENDXREF;"/>

</UMLModelElement.taggedValue>

</UML:Attribute>

Descripción:

A este tipo de dato se lo define independientemente de su uso, es decir, se lo hace aparte de la definición de los elementos que lo utilizan. Por otro lado, su uso se define dentro de la definición de los elementos que lo utilizan.

* ***Tipo de dato cbhdmDateTime***

Definición:

<UML:DataTypexmi.id="**eaxmiid4**" name="**cbhdmDateTime**" isAbstract="**false**" isLeaf="**false**" isRoot="**false**" visibility="**private**"/>

Uso:

```
<UML:Attribute name="FechaCreacion" changeable="none" visibility="private"
ownerScope="instance" targetScope="instance">
        <UML:Attribute.initialValue>
        /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
   <UML:ModelElement.taggedValue>
        <UML:TaggedValue tag="type" value="cbhdmDateTime"/>
              <UML:TaggedValue tag="derived" value="0"/>
              /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
        </UML:ModelElement.taggedValue>
        </UML:Attribute>
```

Descripción:

La definición de este tipo de dato se hace aparte de la definición de los elementos que lo utilizan, pero su uso se define dentro de dichos elementos.

C. Valores de Enumeración

∗ ***Valor de Enumeración "Pendiente"***

Código:

```
<UML:Attribute name="Pendiente" changeable="none" visibility="private"
ownerScope="instance" targetScope="instance">
        <UML:Attribute.initialValue>
              <UML:Expression/>
        </UML:Attribute.initialValue>
        /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
   <UML:ModelElement.stereotype>
        <UML:Stereotype name="enum"/>
   </UML:ModelElement.stereotype>
   <UML:ModelElement.taggedValue>
          <UML:TaggedValue tag="type" value="char"/>
              <UML:TaggedValue tag="derived" value="0"/>
                   <UML:TaggedValuetag="containment"value="Not
Specified"/>      /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
                        </UML:ModelElement.taggedValue>
                        </UML:Attribute>
```

Descripción:

Al valor de enumeración “Pendiente” se lo define como un atributo, indicando que emplea un estereotipo “enum” (indicando que es una enumeración), y describiendo además a aquellos valores que lo describen (ya sean tipo de dato, si es derivada, etc).

* ***Valor de Enumeración “Rechazado”***

Código:

```
<UML:Attribute name="Rechazada" changeable="none" visibility="private"
ownerScope="instance" targetScope="instance">
            <UML:Attribute.initialValue>
            <UML:Expression/>
</UML:Attribute.initialValue>
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
<UML:ModelElement.stereotype>
<UML:Stereotype name="enum"/>
</UML:ModelElement.stereotype>
<UML:ModelElement.taggedValue>
            <UML:TaggedValue tag="type" value="char"/>
            <UML:TaggedValue tag="derived" value="0"/>
            /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
</UML:ModelElement.taggedValue>
</UML:Attribute>
```

Descripción:

Al valor de enumeración “Rechazado” se lo define como un atributo, indicando que emplea un estereotipo “enum” (indicando que es una enumeración).

* ***Valor de enumeración “Aprobado”***

Código:

```
<UML:Attribute name="Aprobado" changeable="none" visibility="private"
ownerScope="instance" targetScope="instance">

  <UML:Attribute.initialValue>
  <UML:Expression/>
```

```
</UML:Attribute.initialValue>
    \/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\

<UML:ModelElement.stereotype>

<UML:Stereotype name="enum"/>
</UML:ModelElement.stereotype>
<UML:ModelElement.taggedValue>
        <UML:TaggedValue tag="type" value="char"/>
        <UML:TaggedValue tag="derived" value="0"/>
        <UML:TaggedValue tag="containment"value="Not Specified"/>
      <UML:TaggedValue tag="ordered" value="0"/>
              <UML:TaggedValuetag="stereotype"value="enum"/>
                    /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
    </UML:ModelElement.taggedValue>
    </UML:Attribute>
```

Descripción:

Al valor de enumeración “Rechazado” se lo define como un atributo, indicando que emplea un estereotipo “enum” (indicando que es una enumeración). Por otro lado, se incluye una descripción de los datos que lo describen, tales como el tipo de valor (char, int, entre otros), estereotipo que emplea (“enum”, el cual fue mencionado anteriormente), entre varios.

D. Valores Etiquetados

* ***Valor etiquetado “NombreUsuario”***

Código:

```
<UML:TaggedValue
xmi.id="EAID_13124A90_24EA_49ff_938F_806D29321FFE"
value="NombreUsuario" tag="User"
modelElement="EAID_CE980939_A100_4926_B5C7_5339CE32EDB4"/>
```

Descripción:

La definición de este valor etiquetado se realiza de forma independiente, o sea, “fuera” de la definición de aquellos atributos, clases y operaciones que lo utilizan.

* ***Valor etiquetado "Clave"***

Código:

<UML:TaggedValue xmi.id="**EAID_7250D552_EA39_4dcb_932B_AEFAC95511C9**" value="**Clave**" tag="**Password**" modelElement="**EAID_CE980939_A100_4926_B5C7_5339CE32EDB4**"/>

Descripción:

La definición de este valor etiquetado se realiza de forma independiente, o sea, "fuera" de la definición de aquellos atributos, clases y operaciones que lo utilizan.

* ***Valor etiquetado "Ingreso al Sistema"***

Código:

<UML:TaggedValue xmi.id="**EAID_33105616_4D41_4f99_8D2E_3D10B6A46AE6**" value="**Ingreso al Sistema**" tag="**Title**" modelElement="**EAID_CE980939_A100_4926_B5C7_5339CE32EDB4**"/>

Descripción:

La definición de este valor etiquetado, al igual que como sucede con los otros dos valores, se realiza de forma independiente, o sea, "fuera" de la definición de aquellos atributos, clases y operaciones que lo utilizan.

3.3.4.4.2. Xmi 2.1 (Enterprise Architect)

A. **Estereotipos**

* ***Estereotipo "Descriptor"***

Código:

packagedElement name="**Descriptor**" xmi:type="**uml:Stereotype**" xmi:id="**Descriptor**">

<ownedAttribute name="**base_Attribute**" xmi:type="**uml:Property**" xmi:id="**Descriptor-base_Attribute**" association="**Attribute_Descriptor**">

<type href="**http://schema.omg.org/spec/UML/2.1/uml.xml#Attribute**"/>

</ownedAttribute>

</packagedElement>

Implementación:

<attributes>

<attribute name="**Descripcion**" xmi:idref="**EAID_96E645DC_C36D_4639_A4A9_9A128BC58DD7**" scope="**Private**">

<initial/><documentation/><model ea_localid="**9**" ea_guid="**{96E645DC-C36D-4639-A4A9-9A128BC58DD7}**"/>

/\

<stereotype stereotype="**Descriptor**"/>

/\

<xrefs value="**$XREFPROP=$XID={06ED7722-9398-4d33-9EB9-C839F7CA5235}$XID;$NAM=Stereotypes$NAM;$TYP=attribute property$TYP;$VIS=Public$VIS;$DES=@STEREO;Name=Descriptor;GUID={4B3236F1-0B95-406f-81E6-D2375E26DC84};@ENDSTEREO;$DES;$CLT={96E645DC-C36D-4639-A4A9-9A128BC58DD7}$CLT;$SUP=<none>$SUP;$ENDXREF;**"/>

</attribute>

Descripción:

El estereotipo “Descriptor” se define de forma individual, pero, al implementarlo, se lo utiliza en el código correspondiente a la clase que lo emplea.

* ***Estereotipo “Identifier”***

Código:

<packagedElement name="**Identifier**" xmi:type="**uml:Stereotype**" xmi:id="**Identifier**">

<ownedAttribute name="**base_Attribute**" xmi:type="**uml:Property**" xmi:id="**Identifier-base_Attribute**" association="**Attribute_Identifier**">

<type href="**http://schema.omg.org/spec/UML/2.1/uml.xml#Attribute**"/>

</ownedAttribute>

</packagedElement>

Implementación:

attribute name="**IdEstado**"
xmi:idref="**EAID_47EB8FC8_8CEA_43e5_9D0A_1EDCEE681905**"
scope="**Private**"><initial/><documentation/>

/\

<stereotype stereotype="**Identifier**"/>

/\

<xrefs value="**$XREFPROP=$XID={2DB02C25-A710-4246-9D57-875996835FED}$XID;$NAM=Stereotypes$NAM;$TYP=attribute property$TYP;$VIS=Public$VIS;$DES=@STEREO;Name=Identifier;GUID={690C6F40-B808-498f-B9D2-85955051C335};@ENDSTEREO;$DES;$CLT={47EB8FC8-8CEA-43e5-9D0A-1EDCEE681905}$CLT;$SUP=<none>$SUP;$ENDXREF;**"/></attribute>

Descripción:

En este caso, al estereotipo “Identifier” se lo define independientemente de donde se lo use. Como se puede ver, su uso se determina en el código del elemento que lo utiliza (en este caso el atributo “IdEstado” de la clase “Estado”).

B. Tipos de datos

* ***Tipo de dato “cbhdmString”***

Código:

<operation name="**Titulo**"
xmi:idref="**EAID_AF90725C_ACB7_4807_9684_172F379A2BF3**"
scope="**Public**">

<properties position="**2**"/>

<stereotype/>

<model ea_localid="**1**" ea_guid="**{AF90725C-ACB7-4807-9684-172F379A2BF3}**"/>

<type type="**cbhdmString**" concurrency="**Sequential**" isAbstract="**false**" isQuery="**false**" pure="**0**" returnarray="**1**" synchronised="**0**" static="**false**" const="**false**"/>

<behaviour/>

/\

<parameters><parameter visibility="**public**" xmi:idref="**EAID_RETURNID_ACB7_4807_9684_172F379A2BF3**">

<properties type="**cbhdmString**" ea_guid="**{RETURNID-ACB7-4807-9684-172F379A2BF3}**" const="**false**" array="**true**" pos="**0**"/>

/\

</parameter>

/\

</parameters>

<xrefs/>

</operation>

Descripción:

A este tipo de dato se lo define dentro de los elementos que lo emplean, como así también se definen sus características.

* ***Tipo de dato "cbhdmInt"***

Código:

<operation name="**IdTarea**" xmi:idref="**EAID_0C11ABEE_8994_4f23_98CF_B48CFE2CB937**" scope="**Public**">

<properties position="**1**"/>

<stereotype/>

<model ea_localid="**2**" ea_guid="**{0C11ABEE-8994-4f23-98CF-B48CFE2CB937}**"/>

<type type="**cbhdmInt**" concurrency="**Sequential**" isAbstract="**false**" isQuery="**false**" pure="**0**" returnarray="**0**" synchronised="**0**" static="**false**" const="**false**"/>

<behaviour/>

/\

<documentation/>

<tags/>

<parameters>

<parameter visibility="**public**" xmi:idref="**EAID_RETURNID_8994_4f23_98CF_B48CFE2CB937**">

<properties type="**cbhdmInt**" ea_guid="**{RETURNID-8994-4f23-98CF-B48CFE2CB937}**" const="**false**" pos="**0**"/>

</parameter>

/\

</parameters><xrefs/></operation>

Descripción:

El tipo de dato “cbhdmInt” se define dentro del código correspondiente a los elementos que lo utilizan, como así también se le definen sus características.

* ***Tipo de dato “cbhdmDateTime”***

Código:

operation name="**FechaCreacion**" xmi:idref="**EAID_9CE1AE11_31E6_4211_A9EB_F09E912B91E1**" scope="**Public**">

<properties position="**0**"/>

<stereotype/>

<model ea_localid="**3**" ea_guid="**{9CE1AE11-31E6-4211-A9EB-F09E912B91E1}**"/>

<type type="**cbhdmDateTime**" concurrency="**Sequential**" isAbstract="**false**" isQuery="**false**" pure="**0**" returnarray="**0**" synchronised="**0**" static="**false**" const="**false**"/>

<behavior>

/\

<parameters><parameter visibility="**public**" xmi:idref="**EAID_RETURNID_31E6_4211_A9EB_F09E912B91E1**">

<properties type="**cbhdmDateTime**" ea_guid="**{RETURNID-31E6-4211-A9EB-F09E912B91E1}**" const="**false**" pos="**0**"/>

/\

</ parameter>

</parameters><xrefs/></operation>

Descripción:

Al igual que con lo que sucede con los otros tipos de datos, a “cbhdmDateTime” se lo define dentro del código correspondiente al elemento (en este caso, al atributo “Fecha Creación”) que lo utiliza.

C. Valores de enumeración

* ***Valor de enumeración “Pendiente”***

Código:

<ownedAttribute visibility="**private**" name="**Pendiente**" xmi:type="**uml:Property**" xmi:id="**EAID_635142D1_9C0E_4c7e_843A_7D5E9B861D58**" isDerived="**false**">

<lowerValue xmi:type="**uml:LiteralInteger**" xmi:id="**EAID_LI000007_9C0E_4c7e_843A_7D5E9B861D58**" value="**1**"/>

<upperValue xmi:type="**uml:LiteralInteger**" xmi:id="**EAID_LI000008_9C0E_4c7e_843A_7D5E9B861D58**" value="**1**"/>

</ownedAttribute>

Descripción:

El valor de enumeración “Pendiente” se define independientemente del elemento que lo utiliza (se lo define “fuera” del código respectivo al/los elementos/s que lo utiliza/n).

* ***Valor de enumeración “Aprobado”***

Código:

ownedAttribute visibility="**private**" name="**Aprobado**" xmi:type="**uml:Property**" xmi:id="**EAID_9F2FCD2F_B608_47ac_AC40_1A1BC7F97FE7**" isDerived="**false**">

<lowerValue xmi:type="**uml:LiteralInteger**" xmi:id="**EAID_LI000005_B608_47ac_AC40_1A1BC7F97FE7**" value="**1**"/>

<upperValue xmi:type="**uml:LiteralInteger**" xmi:id="**EAID_LI000006_B608_47ac_AC40_1A1BC7F97FE7**" value="**1**"/>

</ownedAttribute>

Descripción:

El valor de enumeración “Aprobado” se define de forma individual respecto al elemento que los utiliza (se lo define fuera del código de dicho elemento).

* ***Valor de enumeración “Rechazado”***

Código:

ownedAttribute visibility="**private**" name="**Rechazada**" xmi:type="**uml:Property**" xmi:id="**EAID_6FD5F7A2_4D1E_4fd4_897A_3069E3D255B2**" isDerived="**false**">

<lowerValue xmi:type="**uml:LiteralInteger**" xmi:id="**EAID_LI000009_4D1E_4fd4_897A_3069E3D255B2**" value="**1**"/>

<upperValue xmi:type="**uml:LiteralInteger**" xmi:id="**EAID_LI000010_4D1E_4fd4_897A_3069E3D255B2**" value="**1**"/>

</ownedAttribute>

Descripción:

De forma similar a lo que sucede con los valores “Aprobado” y “Pendiente”, el valor de enumeración “Rechazado” se define de forma individual respecto al/los elemento/s que lo emplea/n.

D. Valores etiquetados

* ***Valor Etiquetado “nombreUsuario”***

Código:

<element name="**Componente**" xmi:type="**uml:Component**" xmi:idref="**EAID_CE980939_A100_4926_B5C7_5339CE32EDB4**" scope="**public**">

/\

<tags>

/\

<tag name="**User**" xmi:id="**EAID_13124A90_24EA_49ff_938F_806D29321FFE**" value="**NombreUsuario**" modelElement="**EAID_CE980939_A100_4926_B5C7_5339CE32EDB4**"/></tags>

/\

</element>

Descripción:

El valor etiquetado “nombreUsuario” se define dentro del código que corresponde al elemento que lo utiliza (en este caso, la clase “Componente”).

* ***Valor etiquetado “Clave”***

Código:

<element name="**Componente**" xmi:type="**uml:Component**" xmi:idref="**EAID_CE980939_A100_4926_B5C7_5339CE32EDB4**" scope="**public**">

/\

<tags>

/\

<tag name="**Password**" xmi:id="**EAID_7250D552_EA39_4dcb_932B_AEFAC95511C9**" value="**Clave**" modelElement="**EAID_CE980939_A100_4926_B5C7_5339CE32EDB4**"/>

/\

</element>

Descripción:

Este valor etiquetado se define dentro del código que corresponde a la clase que lo utiliza.

* ***Valor Etiquetado "Ingreso al Sistema"***

Código:

<element name="**Componente**" xmi:type="**uml:Component**" xmi:idref="**EAID_CE980939_A100_4926_B5C7_5339CE32EDB4**" scope="**public**">

/\

<tags>

/\

<tag name="**Title**" xmi:id="**EAID_33105616_4D41_4f99_8D2E_3D10B6A46AE6**" value="**Ingreso al Sistema**" modelElement="**EAID_CE980939_A100_4926_B5C7_5339CE32EDB4**"/>

/\

</element>

Descripción:

Este valor etiquetado, al igual con lo que sucede con los valores "Clave" y "nombreUsuario", se define dentro del elemento (en este caso, la clase "Componente") que lo utiliza.

3.3.4.4.3. Xmi 2.1 (Magic Draw)

A. Estereotipos

* ***Descriptor***

Código:

<nestedClassifierxmi:type='uml:Stereotype'
xmi:id='_17_0_4_e70033f_1372683471235_959805_3397'
name='Descriptor'>

<ownedAttributexmi:type='uml:Property'
xmi:id='_17_0_4_e70033f_1372683471235_300593_3399'
name='base_Operation'visibility='private'
association='_17_0_4_e70033f_1372683471235_102132_3398'>

<type href='http://www.omg.org/spec/UML/20110701/UML.xmi#Operation'>

<xmi:Extension extender='MagicDraw UML 17.0.4'>

<referenceExtension referentPath='UML Standard Profile::UML2 Metamodel::Operation' referentType='Class'/>

</xmi:Extension>

</type>

</ownedAttribute>

</nestedClassifier>

Descripción:

El estereotipo “Descriptor” se define en el código Xmi de forma “independiente”, o sea, se lo define fuera del código que corresponde a la definición de aquellos elementos (tales como clases, atributos, entre otros) que emplean su uso.

* ***Identifier***

Código:

<nestedClassifier xmi:type='uml:Stereotype'
xmi:id='_17_0_4_e70033f_1372683003462_113539_3350' name='Identifier'>

<ownedAttribute xmi:type='uml:Property'
xmi:id='_17_0_4_e70033f_1372683003463_901973_3352'

name='base_Property' visibility='private'
association='_17_0_4_e70033f_1372683003463_670683_3351'>

<type href='http://www.omg.org/spec/UML/20110701/UML.xmi#Property'>

<xmi:Extension extender='MagicDraw UML 17.0.4'>

<referenceExtension referentPath='UML Standard Profile::UML2 Metamodel::Property' referentType='Class'/>

</xmi:Extension>

</type>

</ownedAttribute>

</nestedClassifier>

Descripción:

Al estereotipo “Identifier” se lo define en una porción de código que se encuentra fuera de la parte que corresponde a la definición de aquellos elementos (tales como clases, atributos, entre otros) que lo utilizan.

B. Tipos de Datos

* ***Tipo de dato cbhdmString***

Código:

packagedElement name="**Tareas**"
xmi:id="**_17_0_4_e70033f_1372682523302_398953_3287**"
xmi:type="**uml:Class**">

/\

<nestedClassifier name="**cbhdmString**"
xmi:id="**_17_0_4_e70033f_1372682583660_849858_3309**"
xmi:type="**uml:DataType**"/>

/\

</packagedElement>

Descripción:

Al tipo de dato “cbhdmString” se lo define dentro de la clase que lo utiliza (en este caso, la clase “Tareas”).

* ***Tipo de dato cbhdmInt***

Código:

packagedElement name="**Tareas**"
xmi:id="**_17_0_4_e70033f_1372682523302_398953_3287**"
xmi:type="**uml:Class**">

/\

<nestedClassifier name="**cbhdmInt**"
xmi:id="**_17_0_4_e70033f_1372682876790_240529_3332**"
xmi:type="**uml:DataType**"/>

/\

</packagedElement>

Descripción:

El tipo de dato “cbhdmInt” es definido dentro de la clase “Tareas”, la cual es la única clase que lo emplea (en este diagrama).

* ***Tipo de dato cbhdmDateTime***

Código:

<packagedElement name="**Tareas**"
xmi:id="**_17_0_4_e70033f_1372682523302_398953_3287**"
xmi:type="**uml:Class**">

/\

<nestedClassifier name="**cbhdmDateTime**"
xmi:id="**_17_0_4_e70033f_1372683222847_901995_3366**"
xmi:type="**uml:DataType**"/>

/\

</packagedElement>

Descripción:

El tipo de dato “cbhdmDateTime” se define dentro de la clase que lo emplea (“Tareas”).

C. Valores de Enumeración

* ***Valor de Enumeración "Pendiente"***

Código:

<packagedElement name="**Estados**"
xmi:id="**_17_0_4_e70033f_1372684668038_918898_3721**"
xmi:type="**uml:Enumeration**">

/\

<ownedLiteral name="**Pendiente**"
xmi:id="**_17_0_4_e70033f_1372684781856_420109_3752**"
xmi:type="**uml:EnumerationLiteral**"/>

/\

</packagedElement>

Descripción:

El valor de enumeración "Pendiente" está definido dentro de la clase "Estados", la cual lo emplea como tal.

* ***Valor de Enumeración "Aprobada"***

Código:

<packagedElement name="**Estados**"
xmi:id="**_17_0_4_e70033f_1372684668038_918898_3721**"
xmi:type="**uml:Enumeration**">

/\

<ownedLiteral name="**Aprobado**"
xmi:id="**_17_0_4_e70033f_1372684797557_461704_3754**"
xmi:type="**uml:EnumerationLiteral**"/>

/\

</packagedElement>

Descripción:

El valor de enumeración "Aprobada" se define dentro de la clase "Estados", siendo esta la clase que lo utiliza.

* ***Valor de Enumeración "Rechazada"***

Código:

```
<packagedElement name="Estados"
xmi:id="_17_0_4_e70033f_1372684668038_918898_3721"
xmi:type="uml:Enumeration">
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
<ownedLiteral name="Rechazado"
xmi:id="_17_0_4_e70033f_1372684803549_801488_3756"
xmi:type="uml:EnumerationLiteral"/>
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
</packagedElement>
```

Descripción:

El valor de enumeración "Rechazado" se define dentro de la clase donde se aplica (clase "Estados").

D. **Valores Etiquetados**

* ***Valor Etiquetado "NombreUsuario"***

Código:

```
packagedElement name="Component"
xmi:id="_17_0_4_e70033f_1372685501497_837341_3790"
xmi:type="uml:Artifact">
<ownedAttribute name="NombreUsuario"
xmi:id="_17_0_4_e70033f_1372689874124_614532_3883"
xmi:type="uml:Property" visibility="private"/>
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
</packagedElement>
```

Descripción:

A este valor etiquetado se lo define dentro del código que corresponde a la clase que lo emplea (Clase "Component").

* ***Valor Etiquetado "Clave"***

Código:

packagedElement name="**Component**"
xmi:id="**_17_0_4_e70033f_1372685501497_837341_3790**"
xmi:type="**uml:Artifact**">

/\

<ownedAttribute name="**Clave**"
xmi:id="**_17_0_4_e70033f_1372691056361_541827_3908**"
xmi:type="**uml:Property**" visibility="**private**"/>

/\

</packagedElement>

Descripción:

Al igual que al valor etiquetado "nombreUsuario", el valor etiquetado "Clave" está definido en el código de la clase "Component", siendo esta la clase que lo emplea.

* ***Valor etiquetado "Ingreso al sistema"***

Código:

packagedElement name="**Component**"
xmi:id="**_17_0_4_e70033f_1372685501497_837341_3790**"
xmi:type="**uml:Artifact**">

/\

<ownedAttribute name="**IngresoAlSistema**"
xmi:id="**_17_0_4_e70033f_1372691085587_97345_3913**"
xmi:type="**uml:Property**" visibility="**private**"/>

/\

</packagedElement>

Descripción:

De forma similar que a los valores etiquetados de "nombreUsuario" y "clave", el valor "Ingreso al sistema" se define en el código que corresponde a la clase "component".

3.3.5. Etapa 5. – Construcción de una Herramienta.

- ✓ **Definición del BNF – (Backup Naur Form)**

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

Este BNF fue generado incluyendo todas las características de los distintos componentes del sistema y además incluye una sección que debe ser generada en forma dinámica por la herramienta de transformación para poder validar la correcta escritura de entidades y propiedades de las mismas dentro del modelo. La figura 83 muestra un ejemplo de un código BNF que deberá ser generado en forma automática por la herramienta para el caso de un sistema de administración de viajes en taxi.

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

Figura 83. Fragmento de código BNF generado en forma automática para una aplicación tomando como base el modelo conceptual.

- ✓ **Construcción del Parser**

En la Figura 84 se puede observar una captura de la carpeta donde se encuentran otras carpetas que tienen el Parser y las clases generadas. En el Anexo C se muestra el código de la Clase Interoperabilidad.

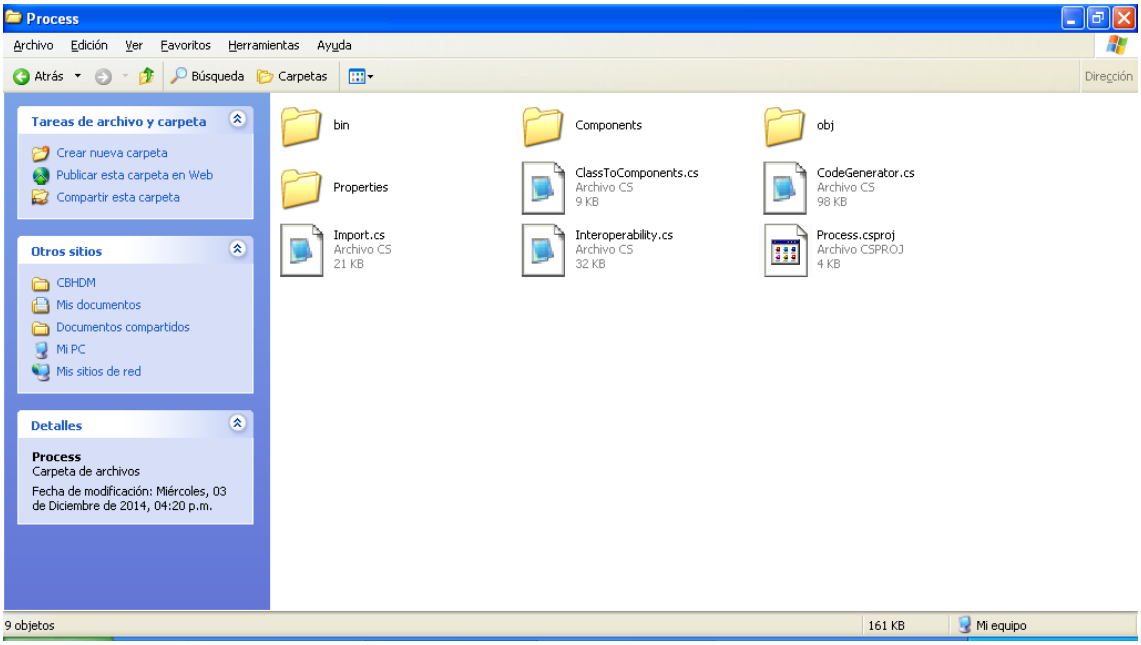


Figura 84. Captura de imágen que muestra las carpetas con el Parser y las clases generadas.

Contenidos de la carpeta COMPONENT, en donde se ven las clases generadas para los distintos tipos componentes que tiene la metodología.

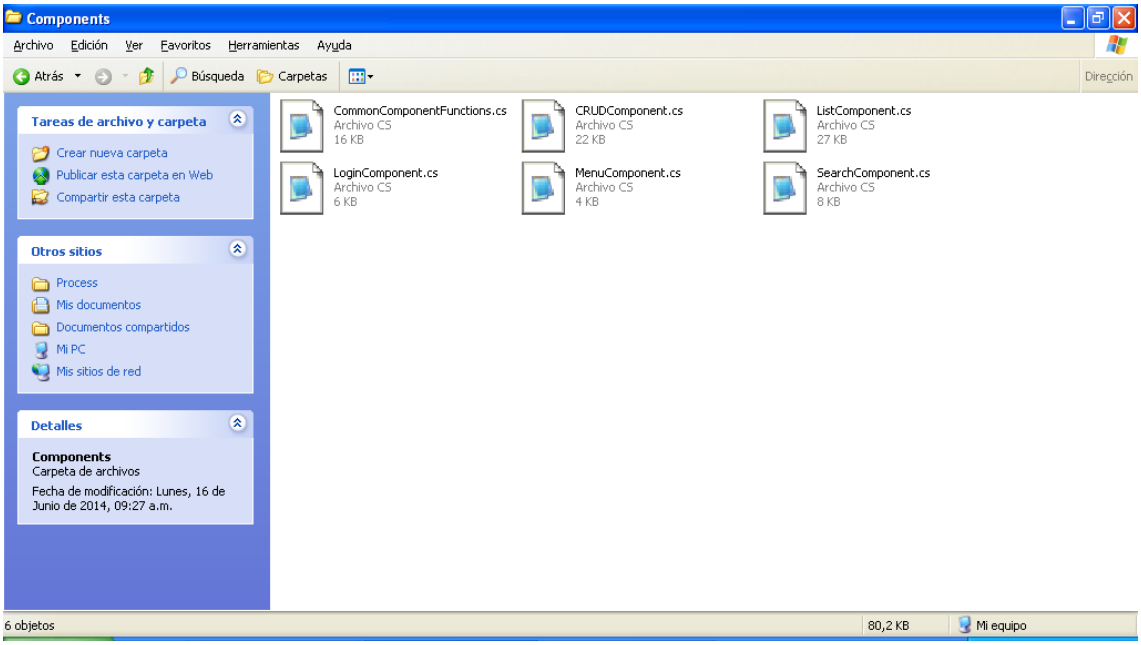
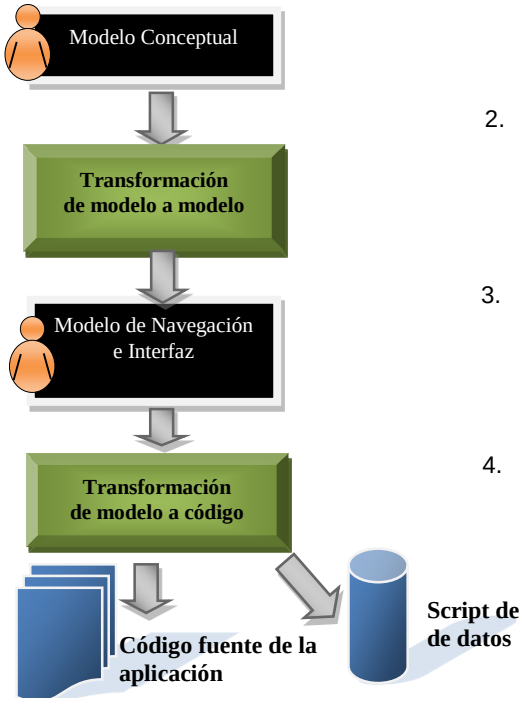


Figura 85. Contenido de la Carpeta COMPONENT.

- ✓ **Transformación entre modelos**

  Esta metodología se enmarca en el enfoque MDA (Arquitectura Definida por Modelos) donde todo el desarrollo está basado en la realización de modelos y mediante transformaciones se va avanzando hasta llegar a generar código fuente. Tal como se mostró en los pasos de la Figura 86 esta metodología incluye dos

transformaciones, la primera de ella de modelo a modelo y la segunda de modelo a código. En la figura 86 se detallan cada una de ellas



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

Figura 86. Pasos de la Metodología.

* ***Transformación de Modelo a Modelo***

Una vez finalizado el modelo conceptual el usuario exporta el diagrama de clases a un archivo XMI [OMG13b] y lo importa en la herramienta de transformación. La herramienta reconoce cada una de las clases y genera un nuevo archivo XMI correspondiente a una primer versión del diagrama de navegación e interfaz que luego el usuario podrá modificar y completar según las necesidades de la aplicación. De forma automática se generan las siguientes transformaciones:

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

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

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

* ***Transformaciones de Modelo a Código***

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

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

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

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

- ✓ Cada propiedad pública de una clase será mapeada a una propiedad en la clase en el lenguaje de programación destino mediante una variable privada y su correspondiente acceso público.
- ✓ Según la navegación se crean las diferentes páginas y vistas con el acceso a datos.
- ✓ Para las pantallas de actualización de datos (CRUD) se derivan los controles de forma automática de acuerdo al tipo de dato, si la relación es con otra entidad se pone un combo de selección con la descripción de la entidad relacionada.

- ✓ Cada uno de los componentes especificados en el modelo de navegación deberá ser renderizado a un control según su funcionalidad.
- ✓ Para las clases del tipo enumeración se creará un tipo de clases de negocio particular donde no tendrá actualización pero si acceso a los datos. A su vez se deberán generar los valores posibles de dicha enumeración en el script de base de datos adicionando los registros correspondientes.

La aplicación web será generada teniendo en cuenta las buenas prácticas para aplicaciones web móviles de la [W3C 10] lo que asegura entre otras cosas optimizar el uso de la red y generar una buena experiencia de usuario.

✓ **Definición de la estructura de los Templates Ejemplo**

Se decide implementar una solución que permita generar código siguiendo un template (plantilla) de aspectos particulares entre ellos opciones rápidas de navegación, con controles predefinidos con un diseño específico, etc. Creandose la solución en HTML5 con JQUERY mobile. Para esto es importante la construcción de una herramienta planificada para poder realizar cambios futuros de forma fácil y poder elegir exportar la solución por ejemplo a XHTML Basic 1.1 y de este modo que funcione para Dispositivos Básicos.

✓ **Armado de un Template de Ejemplo**

En este apartado se muestran las capturas de pantallas de la herramienta realizada.

* ***Pantallas de la Herramienta***

La pantalla de login es la primera que aparece al ejecutar la herramienta. Como todo login espera que se ingrese el usuario y contraseña (ver figura 87).

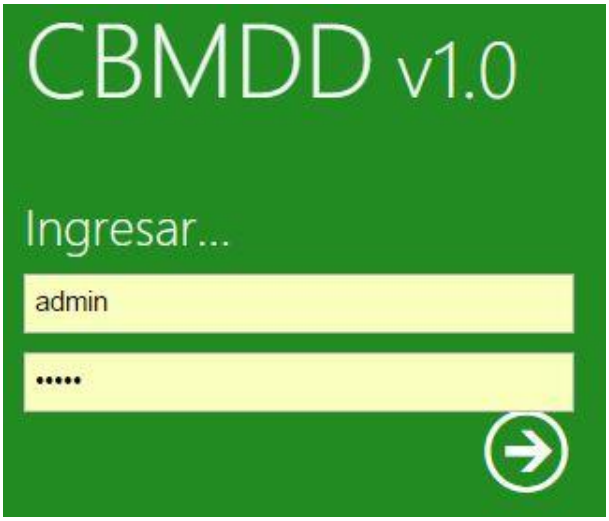


Figura 87. Pantalla de Login

Luego de validar los datos ingresados, el sistema pasará al menú principal (ver figura 88).

Figura 88. Menú Principal

En este se encuentran dos jerarquías principales:

- Proyectos → representado por la opción “Mis Proyectos”, donde se pueden administrar los proyectos generados por la herramienta.

- Usuarios → representado por la opción “Usuarios”, que permite administrar los usuarios del sistema.

En ambas opciones se ve debajo a la derecha un número que representa la cantidad,en el caso de la figura 88 hay 1 proyecto generado y 3 usuarios registrados en el sistema.

Ingresando a proyectos se puede visualizar la pantalla presentada en la figura 89.

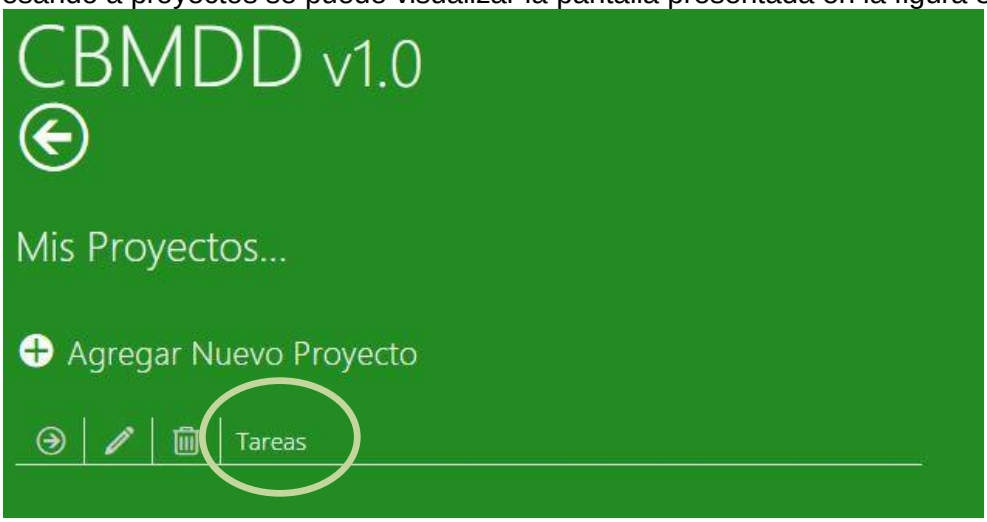


Figura 89. Pantalla Proyectos.

Función: Administra los proyectos.

Crea un nuevo proyecto.


Editar o modificar un proyecto existente.


Eliminar un proyecto.

Administrar las Tareas de un proyecto.

Al ingresar a la opción tareas se visualiza la pantalla presentada en la figura 90.

Figura 90. Menú Tareas

La herramienta desarrollada además de tener implementadas las transformaciones automáticas genera la trazabilidad. En la figura 90 puede observarse el cartel que indica que los Componentes Generados están desactualizados, debido a que se realizaron cambios posteriores a la generación en el modelo que toma como entrada dicha transformación. Esta pantalla es una de las más importantes del sistema ya que tiene el acceso a cada uno de los modelos y a la aplicación final construida.

En la figura 91 se muestra la pantalla relacionada con el diagrama de componentes. En este caso se muestra una lista de diagramas de componentes, pudiendo observarse uno denominado Diagrama Base.

Figura 91. Diagrama de Componentes.

En la pantalla de la figura 92 se muestra como se da de alta un componente.

Figura 92. Configuración del Componente. Alta de Componente.

En la figura 93 puede observarse un listado de componentes y los tipos de componentes en cada caso.

Figura 93. Modificación de Diagrama de Componentes.

En la pantalla de la figura 94 puede observarse como se edita o configura un componente ya creado.

Figura 94. Configuración del Componente. Editar Componente.

En cuanto a las clases, la herramienta también permite su edición en la pantalla de la figura 95 se muestra como se pueden agregar clases.

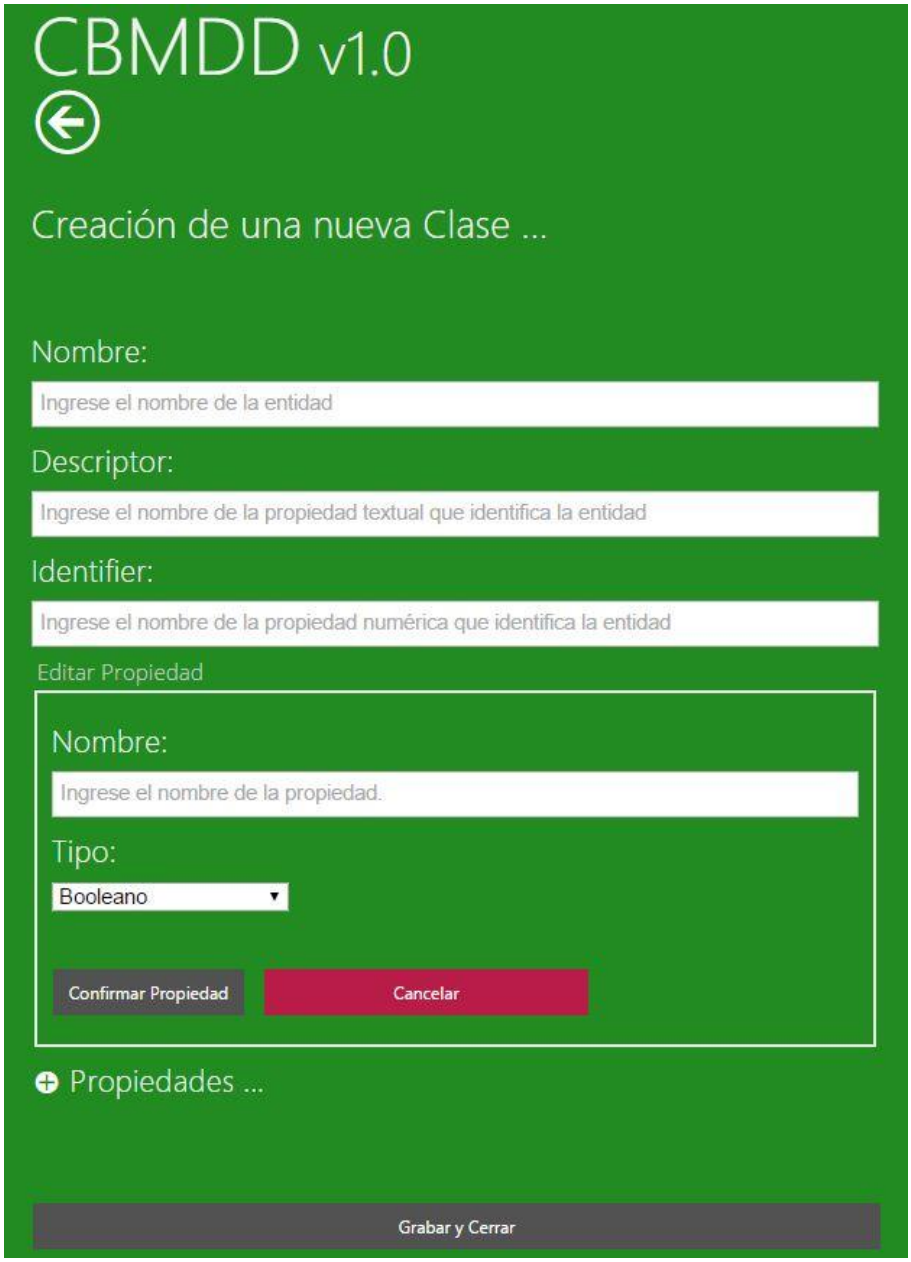


Figura 95. Creación de una nueva Clase.

Función: Crear una nueva clase.

Datos a ingresar:

- Nombre: de la entidad
- Descriptor: nombre de la propiedad textual que identifica la entidad.
- Identifier: Nombre de la propiedad numérica que identifica la entidad
- Editar Propiedades
- Nombre: de la propiedad.
- Tipo: tipo de la propiedad (ejemplo: boolean)

Opciones:

- Confirmar Propiedad
- Cancelar (cancela la propiedad)

Grabar y Cerrar: graba la nueva clase con todos los atributos (datos ingresados) y cierra la pantalla volviendo a la anterior.

Agregar Clase 2

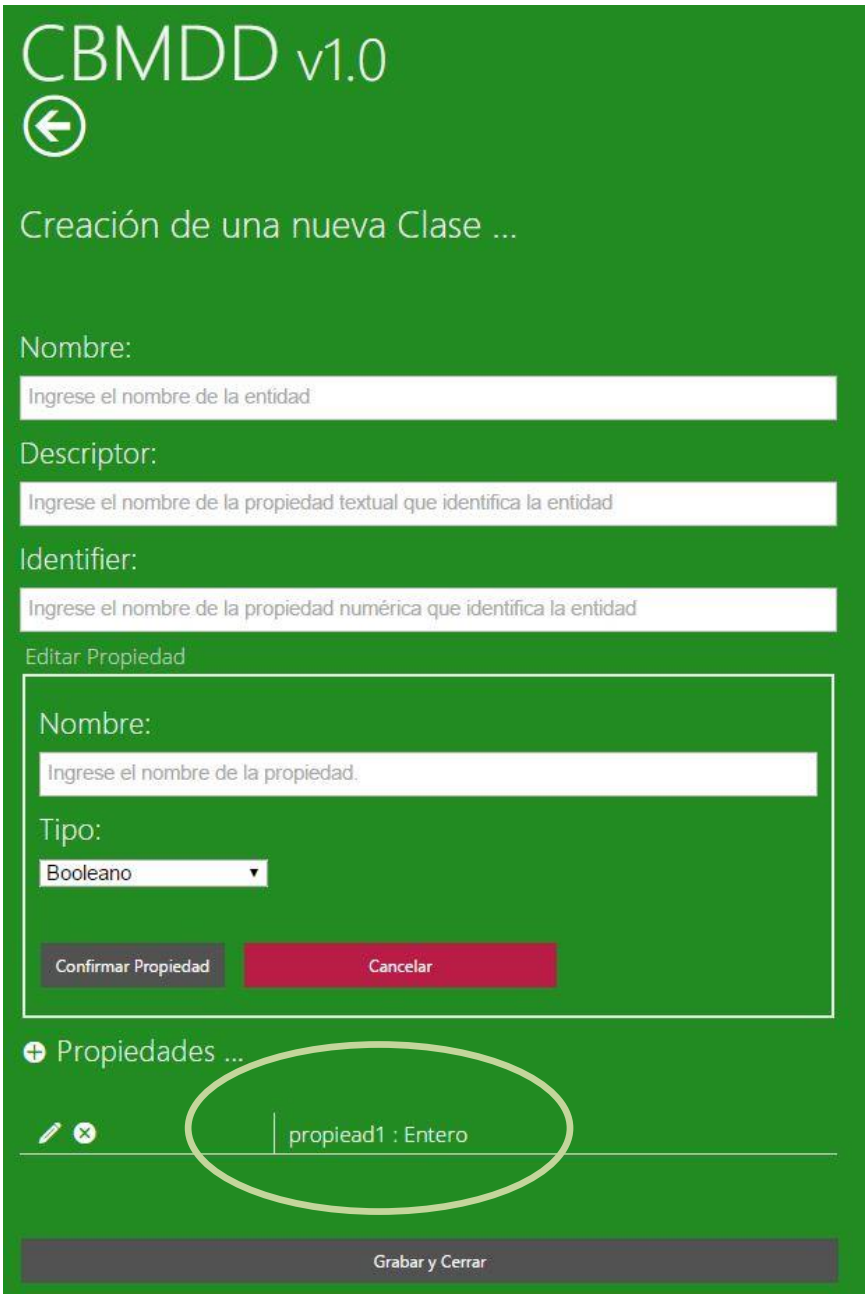


Figura 96. Clase ya creada.

Función: Agregar nuevas propiedades

Como se puede apreciar en lo seleccionado con el ovalo, se listan las propiedades existentes. Las cuales se pueden editar, solo presionando el icono de lápiz.

Ejemplo: Propiedad1: Entero

Propiedad1 (nombre de la propiedad)

Entero (tipo de la propiedad)

La pantalla correspondiente a la figura 97 muestra como es posible administrar las clases ya creadas.

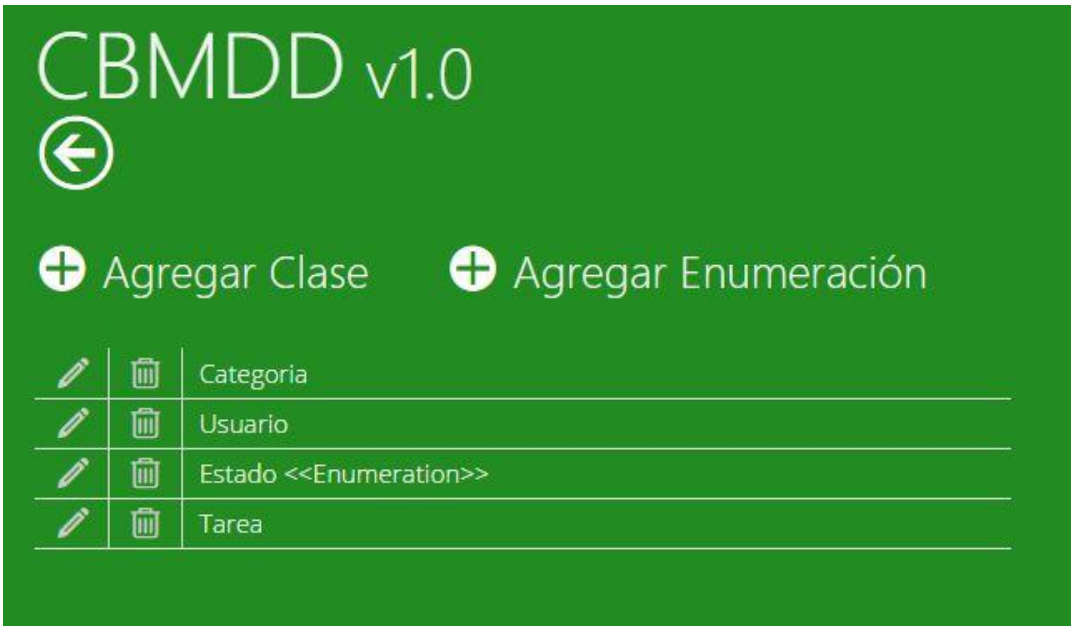


Figura 97. Agregar Clase y Agregar Enumeración.

Función: administrar clases y enumeraciones

Opciones:

- Agregar Clase
- Agregar Enumeración
- Editar → modifica o edita la clase o enumeración seleccionada.
- Eliminar → Elimina una clase o enumeración indicada.

Editar Clase

Figura 98. Modificar Clase.

Función: modificar clase

Una vez seleccionada la clase aparece la siguiente pantalla con los datos propios de la clase (nombre, descriptor e Identifie), y una lista de las propiedades de la clase.

Seleccionando el icono de lápiz de una propiedad puede editarse o modificarse la misma.

En el ejemplo podrían modificarse de la clase “Categoría”, las siguientes propiedades:

- FechaCreación (tipo Fecha/Hora) ,
- FechaModificacion (tipo Fecha/Hora)

- Usuario (tipo Usuario).

Agregar Enumeración

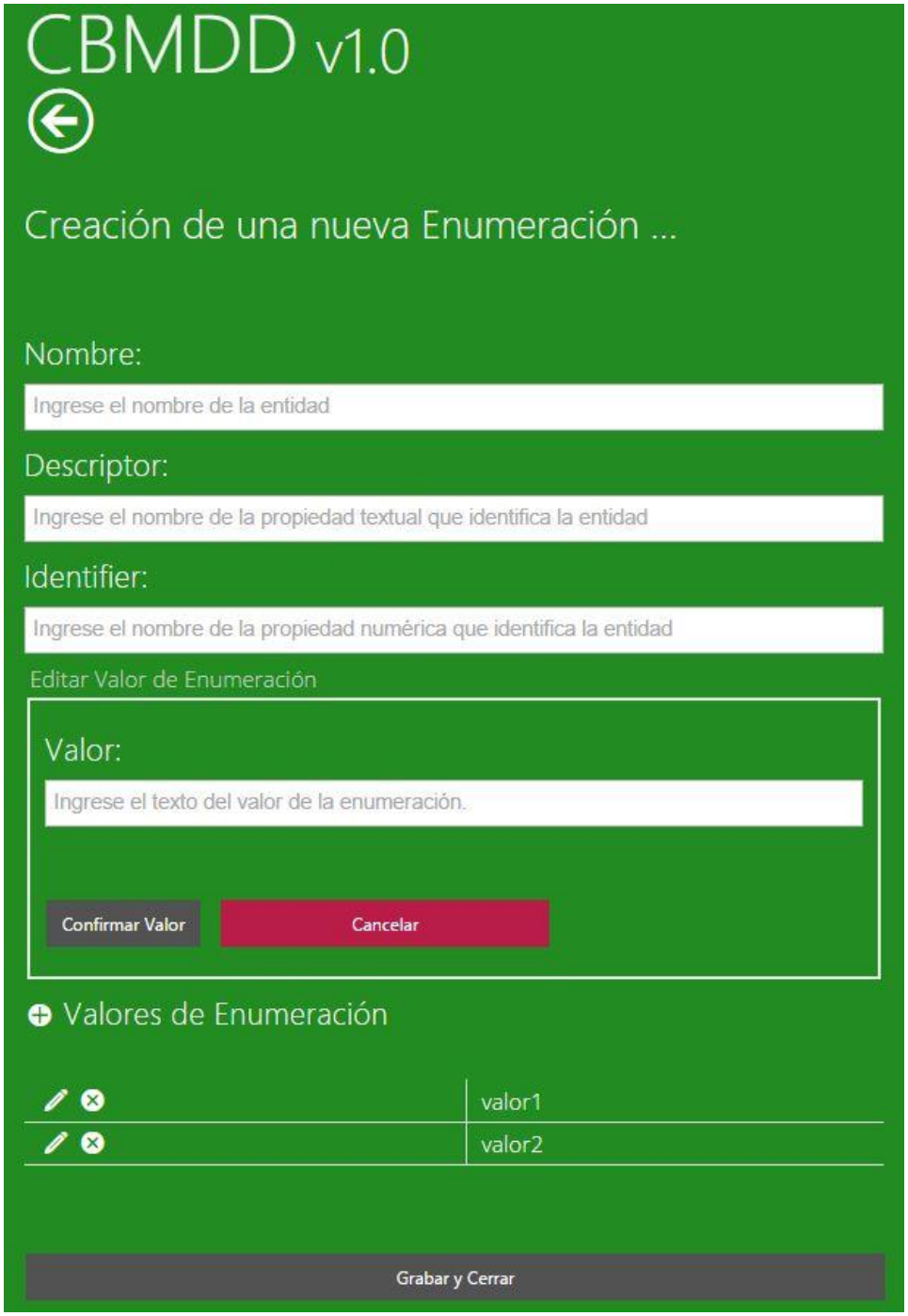


Figura 99. Crear nueva enumeración.

Función: crear una nueva enumeración.

Datos a ingresar.

- Nombre: de la enumeración
- Descriptor: nombre de la propiedad textual que identifica la enumeración.

- Identifier: Nombre de la propiedad numérica que identifica la enumeración.
- Editar Valor de Enumeración: se ingresa el texto del valor de la enumeración.

Opciones:

Grabar y Cerrar: graba la nueva enumeración con todos los valores de la enumeración (datos ingresados) y cierra la pantalla volviendo a la anterior.

Editar Enumeración

Figura 100. Editar Enumeraión.

Función: Editar o modificar una enumeración.

Una vez seleccionada la enumeración y presionado el icono de lápiz, se pasa a esta pantalla, donde aparecen los datos de la enumeración indicada (nombre, descriptor, identifier) Y los valores de la enumeración, para este ejemplo.

- Nombre de la enumeración: Estado
- Descriptor: descriptor
- Identifier: idEstado.

Valores de la enumeración:

- Pendiente
- Finalizada
- Cancelada.

Los cuales se puede modificar presionando el icono de lápiz que corresponde a la enumeración que se desea editar.

Alta de Links

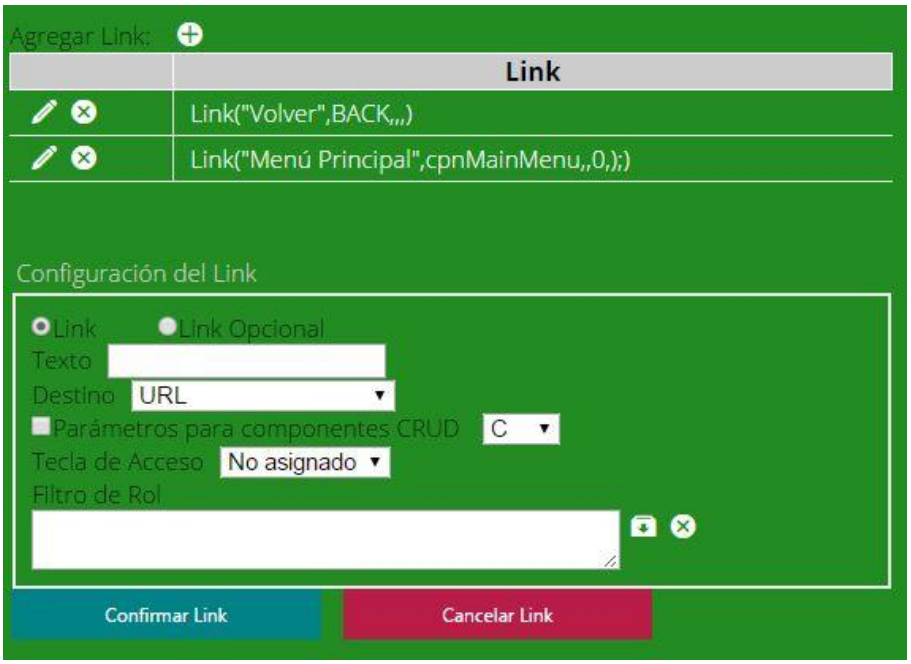


Figura 101. Alta de Links.

Función. Agregar, editar o eliminar Link

Datos a ingresar.

- Link / Link Opcional. Una u otra, opciones excluyentes según corresponda.
- Texto: del link

- Destino: en el ejemplo URL
- Parámetros para componentes: casilla seleccionada o no seleccionada.
- Inicial del componente
- Tecla de acceso. Se puede asignar una tecla para ese link
- Filtro de Rol.

Opciones:

- Confirmar Link: queda grabado el link
- Cancelar: no queda grabado.

Editar Valores Etiquetados

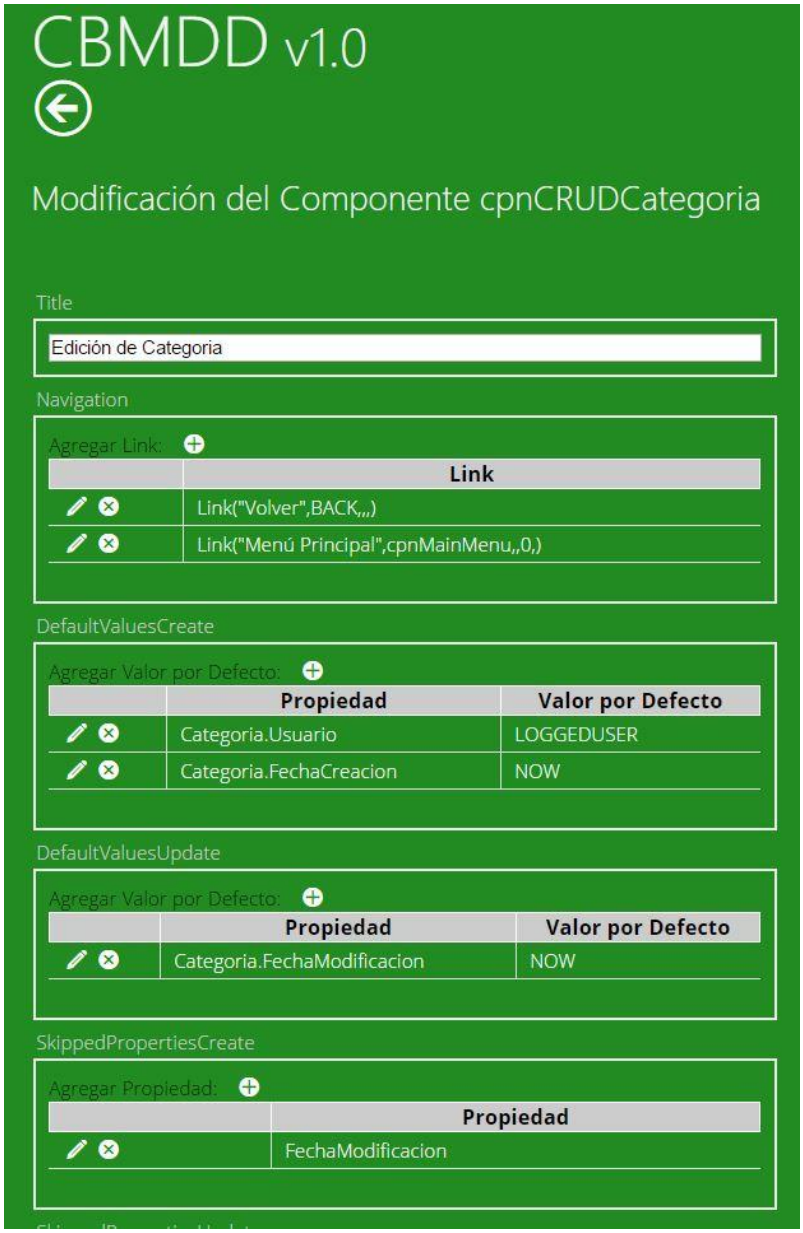


Figura 102. Editar Valores Etiquetados.

Función: Editar componente seleccionado.

Opciones.

- Agregar o editar un link
- Agregar o editar valores por defecto.

Editar Valores Etiquetados de Login

Figura 103. Modificación del Componente cpnLogin.

Función: modificar Componente cpnLogin

Datos a ingresar:

- Title. titulo
- RedirectToComponent,, se elige de una lista desplegable.
- User: se elige de una lista el usuario.
- Password: se deberá ingresar la contraseña.

Opcion.

Grabar y Cerrar. Graba las modificaciones o cancela las mismas

Usuarios del Proyecto

Figura 104. Usuarios del Proyecto.

Función: Habilitar /deshabilitar usuarios.

En la parte izquierda se listan los usuarios Disponibles.

En l aparte derecha los usuarios Habiitados.

Opciones.

Seleccionar un usuario disponible y presionar → el usuario queda en la lista de habilitados.

Con la flecha a la inversa ← se deshabilita el usuario.

Valores por Defecto

Figura 105. Valores por defecto.

Función: agregar Valor por defecto.

Datos a ingresar.

- Elegir Propiedad
- Elegir el valor por defecto.

Opciones:

Confirmar valor por defecto: graba el valor por defecto.

Cancelar valor por defecto, cancela grabación.

- ✓ **Generación a código fuente y generación de la Base de Datos.**
  Modelo de datos se basa en el diagrama de clases UML donde cada clase representará una tabla de base de datos. Este diagrama se amplía con:

  (A) Estereotipos de propiedades especiales: como el identificador único numérico de la tabla y el descriptor textual que da un identificador legible del registro de base de datos.

  (B) Las clases de enumeración: En algunas aplicaciones será necesario identificar aquellas clases que representan valores limitados, como por ejemplo los diferentes estados de una factura, una lista de acciones permitidas, etc. Estas clases deben distinguirse de las otras clases porque deberán tener un tratamiento especial cuando se genera el código fuente. Con el fin de identificar aquellas clases , se utilizará el estereotipo de UML <enumeración> .

  (C) Tipos de datos especiales: Con el fin de mejorar la experiencia del usuario al utilizar la aplicación, sobre todo desde un dispositivo móvil, se utilizarán dos tipos de datos especiales:

  - ***phoneNumber*** que es una cadena que representa un número de teléfono que permite hacer un enlace para automáticamente realizar una llamada telefónica o enviar un SMS a ese número;
  - ***dirección:*** este tipo de datos hace que se puedan utilizar el dispositivo GPS de los teléfonos móviles, y tomar el lugar exacto donde se encuentra el usuario. Si el GPS no está disponible o está temporalmente fuera de servicio, será posible introducir la dirección como una cadena. Este tipo de datos posibilita que se pueda utilizar el GPS para navegar a esa ubicación.

Además la herramienta soporta el agregar clases y enumeraciones; y facilitar el proceso de configuración de las propiedades especiales. En la figura 106 pueden verse imágenes de la herramienta.

Figura 106. Imágenes de la herramienta de soporte al Modelo de Datos.

Después de configurar las clases y enumeraciones, con el uso de la opción "Generar Componentes", el sistema genera automáticamente una versión preliminar del modelo de interfaz de usuario. Pero también con el modelo de datos, un script de base de datos puede ser generado. Esto se puede observar en la figura a continuación.

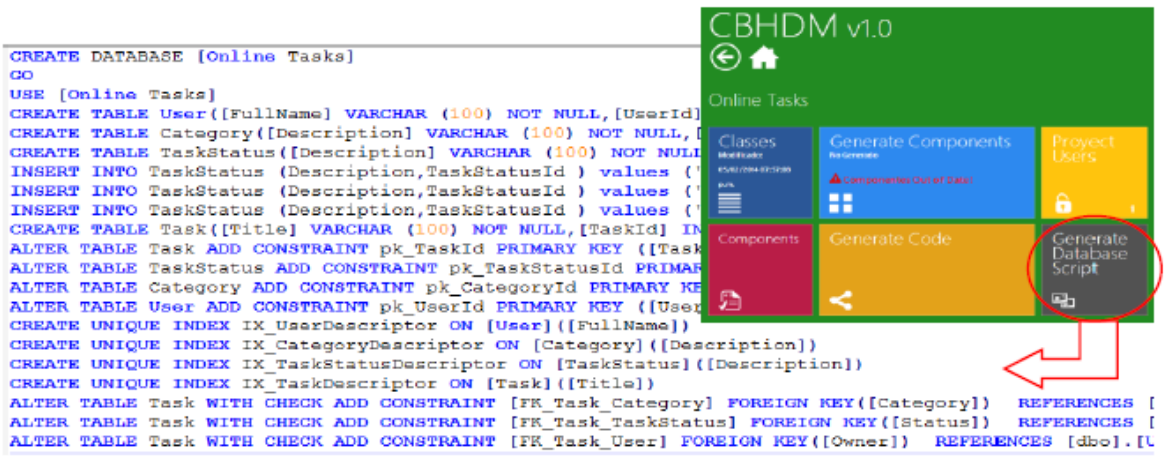


Figura 107. Generando Script de Base de Datos para el Modelo UI.

3.3.6. Etapa 6 – Validación y Pruebas

Habiendo desarrollado la herramienta cuyas capturas fueron presentadas en la sección anterior; se procede a realizar pruebas en cuanto a su funcionalidad corrigiendo errores de programación. Una vez que la herramienta estuvo operativa las Pruebas siguientes consistieron en modelar casos planteados, previamente los cuales habían sido modelados con otras metodologías (etapa 2 y 3 del presente informe), con la herramienta desarrollada analizando las tranformaciones automáticas y el código fuente final generado. Dichos modelos se presentan a continuación en la Etapa 7.

3.3.7. Etapa 7 – Utilización de la Herramienta.

- ✓ **Posibles casos planteados.**

  Se toma en consideración los casos planteados inicialmente y modelados con otras metodologías. A continuación se expone lo realizado para uno de dichos casos que consiste en una aplicación de registración de tareas.

  Enunciado:

  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 es una tarea 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 hacer clic en la descripción el usuario será redireccionado 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 hacer click 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.
- **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 si 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 boolean permitendo seleccionar, la prioritarias, las no prioritaraias 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.

Modelado 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 de acceso al sistema del usuario

- 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 1.

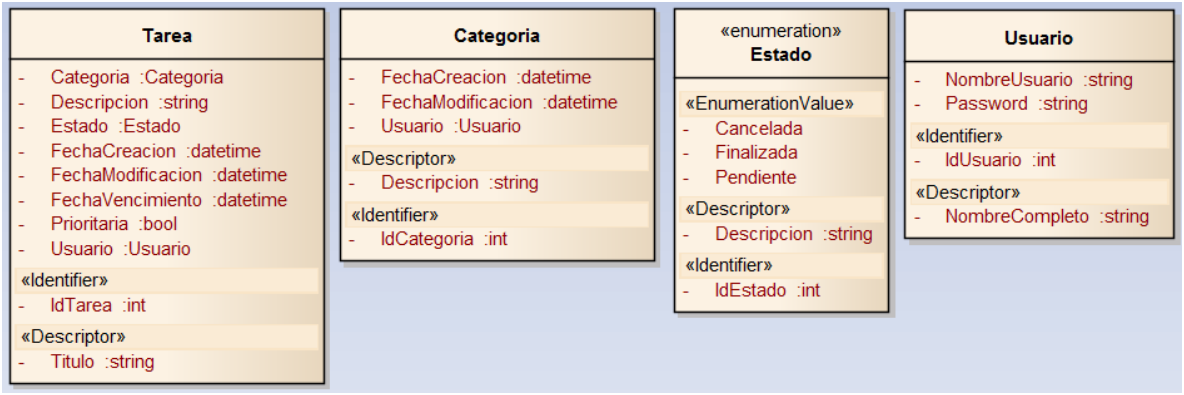


Figura 108. Modelo de datos del sistema de registración de tareas

Modelo de Interfaz de Usuario:

Una vez finalizado el modelo de datos se procede a realizar la primer transformación y se obtiene automáticamente el modelo de interfaz de usuario de la Figura 2.

Figura 2. 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 13.
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 14.
4. Se modifica 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 a 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 15.
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 16.
6. Al componente menú se le agrega un link en la barra de navegación con la opción Logout, que volverá a la pantalla de login. Además se modifican los links para apuntar a los componentes creados agregando además un link para permitir el alta de tareas. La configuración del componente menú puede verse en la Tabla 17.
7. 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 creo el registro en forma automática. La configuración de dichos componentes incluyendo los valores por defecto para cada operación se detallan en la tabla 18 para el caso de las Tareas y en la tabla 19 para las categorías.

Tabla 13. Valores etiquetados del componente de Ingreso al Sistema (tipo `Login`)

Etiqueta	Valor
Id	cpnLogin
Title	Sistema de Registración de Tareas
MainEntity	Usuario
RedirectToComponent	cpnMainMenu
User	NombreUsuario
Password	Password
SelectableUser	False

Tabla 14. Valores etiquetados del listado de tareas prioritarias (componente de tipo `List`)

Etiqueta	Valor
Id	cpnLstTareasPrioritarias
Title	Tareas Prioritarias
MainEntity	Tarea
Navigation	Link(“Menú Principal”, cpnMainMenu,,0,)
RequiresAuthentication	True
FixedFilters	Usuario.IdUsuario = LOGGEDUSER AND Prioritaria = True
Columns	Titulo
AditionalInformationLine	Categoria,FechaVencimiento
Sort	FechaVencimiento ASC
Actions	
DefaultAction	Link("Editar",cpnCRUDTarea,UD,,)

Tabla 15.Valores etiquetados del listado de tareas pendientes (componente de tipo `List`)

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

Tabla 16. Valores etiquetados de la búsqueda de tareas (componente del tipo `Search`)

Etiqueta	Valor
Id	cpnSchBuscarTareas
Title	Búsqueda de Tareas
MainEntity	Tarea
Navigation	Link(“Menú Principal”, cpnMainMenu,,0,);
RequiresAuthentication	True
FixedFilters	Usuario.IdUsuario = 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

Tabla 17. Valores etiquetados del Menú Principal (componente del tipo `Menu`)

Etiqueta	Valor
Id	cpnMainMenu
Title	Sistema de Registración de 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

Tabla 18. Valores etiquetados del componente CRUD de Tareas

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

Tabla 13. 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

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

Título del Diagrama ✔

Diagrama Base

Componentes...

	Identificador	Tipo
	cpnCRUDCategoria	CRUD
	cpnLstCategoria	List
	cpnCRUDTarea	CRUD
	cpnLstTareaPendientes	List
	cpnMainMenu	Menu
	cpnLogin	Login
	cpnLstTareasPrioritarias	List
	cpnSchBuscarTareas	Search

Figura 110. 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 111.

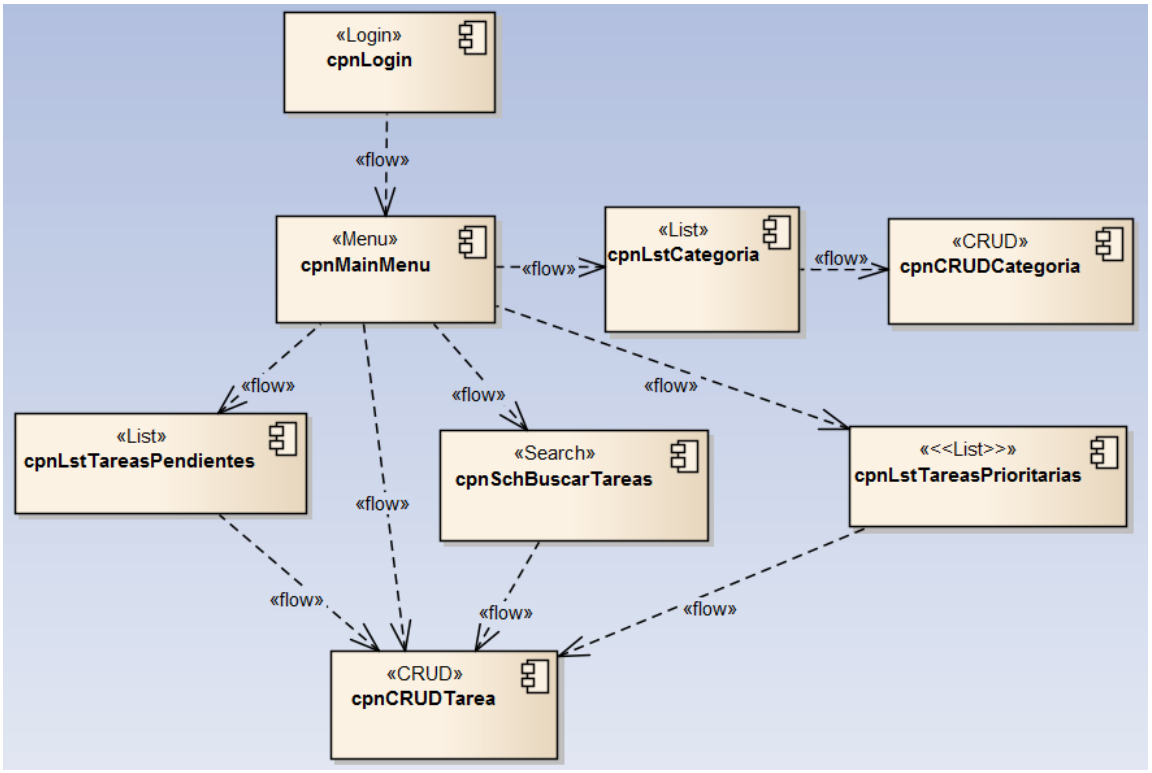


Figura 111. Esquema general de los componentes del sistema

- ✓ ¨**Planteo de nuevos casos posibles.**

  En función de la presentación en congresos académicos y la lectura de material bibliográfico surgieron otros casos a modelar. Uno de ellos pensado para distintos perfiles de usuarios que pueden hacer uso de un mismo dispositivo, por ejemplo en una empresa y como desde la metodología se pueden modelar los roles de usuarios. Fue posible de este modo generar los modelos, aplicar las tranformaciones automáticas y obtener código fuente, probando la herramienta en todos los casos planificados.

4. Resultados

El presente proyecto culmina con una metodología generada que permite aplicar MDA para la generación de aplicaciones móviles. También se desarrollo una herramienta que permite realizar los dos modelos que requiere la metodología y ejecutar las transformaciones automáticas que permiten por un lado generar los script de la base de datos y el código fuente de la aplicación con los menues de navegación, listados, búsquedas y otros controles que son admitidos por la herramienta. Es posible agregar controles para generar interfaces más ricas, también es posible generar otros templates que permitan generar el código fuente por ejemplo para XHTML Basic 1.1 y que sea accesible por dispositivos básicos. Actualmente genera código en HTML 5 con JQUERY Mobile.

Vinculado a este proyecto se encuentra en redacción una tesis doctoral en la Universidad Nacional de La Plata, de uno de los miembros del equipo.

En el apartado siguiente se presentan los artículos expuestos en congresos nacionales e internacionales.

5. Producción Científico-Tecnológica

A continuación se listan las actividades efectuadas en relación con la producción científica vinculada con el presente proyecto (se adjuntan comprobantes de las mismas).

Las que se enumeran a continuación están relacionadas con la metodología planteada en esta línea de Investigación y Desarrollo:

- Congreso Argentino de Ciencias de la Computación (CACIC 2014).
  Título: Automatic Creation of Mobile Web Applications from Design Models.
  Autores: Pablo Vera, Claudia Pons, Carina Gonzalez Gonzalez, Rocío Rodríguez, Daniel Giulianelli.
  Lugar: San Justo, Buenos Aires, Argentina, Octubre 2014.

- Workshop de Investigadores en Ciencias de la Computación (WICC 2014).
  Título: Generación Automática de Aplicaciones Web Móviles Mediante Componentes Configurables.
  Autores: Pablo Vera, Claudia Pons, Carina González González, Rocío Rodríguez, Daniel Giulianelli.
  Lugar: Tierra del Fuego, Argentina. Mayo 2014

- International Conference on Software and Emerging Technologies for Education, Culture, Entertainment, and Commerce (SETECEC 2014).
  ISBN: 978.88.96.471.27.2 - DOI 10.978.8896471/272.
  Título: Tool for Developing Mobile Web Application from UI Models –Based on CBHDM Methodology.
  Autores: Pablo Vera; Claudia Pons, Carina González González, Rocío Rodríguez; Daniel Giulianelli.
  Lugar: Venecia, Italia. Marzo 2014.

- 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.
  Autores: Pablo Vera, Claudia Pons, Carina Gonzales Gonzales, Daniel Giulianelli, Rocío Rodríguez.
  Lugar: Mar del Plata, Buenos Aires, Argentina. Octubre 2013.

- Jornadas Argentinas de Informática e Investigación Operativa (JAIIO 2013).
  Titulo: Metodología de Modelado de Aplicaciones Web Móviles Basada en Componentes de Interfaz de Usuario.
  Autores: Pablo Vera, Claudia Pons, Carina Gonzales Gonzales, Daniel Giulianelli, Rocío Rodríguez.
  Lugar: Córdoba, Argentina. Septiembre 2013.

- Workshop de Investigadores en Ciencias de la Computación (WICC 2013).
  Título: Metodología De Modelado De Aplicaciones Web Móviles Basada En Componentes.
  Autores: Pablo Vera, Claudia Pons, Carina González, Daniel Giulianelli, Rocío Rodríguez.
  Lugar: Paraná, Entre Ríos, Argentina. Abril 2013.

También se efectuó otra publicación, no de la metodología específicamente pero vinculada con el desarrollo web para Dispositivos Móviles:

- Gestión de las TIC's para la Investigación y la Colaboración (TICAL 2013)
  Título: Solución Web para Generar sitios Móviles Accesibles que Permitan Proveer Información Pública Universitaria
  Autores: Artemisa Trigueros, Pablo Vera, Víctor Fernández, Daniel Giulianelli, Rocío Rodríguez, Claudia Alderete
  Lugar: Cartagena de Indias, Colombia[3].

[3] El pasaje Aéreo ha sido financiado por la REDCLARA quienes son parte de la organización del TICAL.

6. Conclusiones

Es importante considerar que el esfuerzo puesto en las primeras etapas del proceso de desarrollo conllevará a un producto final que satisfaga las necesidades del usuario. Pero por otra parte también MDA conlleva a poder generar automáticamente código fuente tomando en consideración los modelos creados. Dichos modelos pueden ser generados con cualquier herramienta case y exportados a XMI un formato estandarizado de intercambio basado en XML, al cual se ha dedicado bastante tiempo de análisis en el primer año del proyecto.

A partir del análisis sobre la exportación, a una misma versión de XMI de un mismo modelo, mediante distintas herramientas pudo observarse que genera resultados distintos no pudiendo importar lo exportado por una herramienta a otra. De hecho en EA (Enterprise Architect) existe la posibilidad de exportar a XMI, lo cual al importarse lo generado en esta misma herramienta tampoco es recuperado correctamente; es por ello que EA incluye un seteo de agregar al XMI características propias del EA y en ese caso sí la herramienta recupera el modelo, pero es un modelo que ya no es interpretado por otras herramientas. Se ha realizado un trabajo arduo para generar modelos en diversas herramientas, exportar a XMI 2.1 en todas ellas y poder analizar las diferencias de aplicación del estándar.

Otra tarea altamente importante ha sido estudiar lenguajes de modelado existentes como UWE, WebML-IFML, OOHDM etc. para poder determinar sus ventajas y desventajas. Decidiendo extender a UML para generar un profile conservativo propio para el dominio de las aplicaciones móviles.

Fue posible generar una nueva metodología denominada CBHDM que cuenta con una herramienta que fue desarrollada con la finalidad de dar soporte al usuario no sólo en el modelado sino también efectuando las transformaciones automáticas y obteniendo el código fuente de la aplicación funcional. Se ha generado un template para HTML 5 con JQUERY, pero la herramienta ha sido planificada para poder incorporar nuevos templates y de este modo poder elegir a que lenguaje exportar el código fuente, es decir, podrían generarse pantallas por ejemplo para dispositivos básicos en XHTML Basic 1.1.

Creemos que lo importante de este trabajo reside en: (1) generar una contribución teórica planteando una metodología, extendiendo de forma conservativa a UML; seleccionar modelos; estereotipos y generar tranformaciones automáticas que permitan obtener el código fuente de una aplicación funcional. (2) desarrollar una herramienta que permita verificar la trazabilidad de los modelos, que tenga interoperabilidad permitiendo importar modelos generados en otra herramienta de modelado (para lo cual se creó un parser del XMI

2.2), que de soporte a la configuración de los componentes y que de forma transparente al usuario, genere los scripts de la base de datos y la aplicación funcional.

7. Bibliografía

[CER00] Ceri S., Fraternali P., Bongio. “Web Modeling Language (WebML): a modeling language for designing Web sites”, Computer Networks, Volume 33, Issues 1–6, (2000), pp 137-157.

[COW95] D. D. Cowan; C. J. P.Lucena, “Abstract Data Views, An Interface Specification Concept to Enhance Design for Reuse”, IEEE Transactions on Software Engineering, Vol.21, No.3, March 1995.

[COW95] Cowan D. and Lucena C.. “Abstract Data Views: An Interface Specification Concept to Enhance Design for Reuse”. IEEE Trans. Softw. Eng. 21, 3 (1995), pp. 229-243.

[DAN11] Marcela Daniele, Ariel Arsaute, Mariana Frutos, Ariel Gonzalez, Paola Martellotto, Marcelo Uva, Fabio Zorzan, Daniel Riesco. “Transformación de modelos aplicada a la definición genérica de casos de uso utilizando QVT y RTG (Reglas de transformación de grafos)”. http://sedici.unlp.edu.ar/handle/10915/20114

[DEF00] Desfray Philipppe. “UML Profile versus Metamodel Extensions: An ongoing debate” (2000). http://www.omg.org/news/meetings/workshops/presentations/uml_presentations/5-3%20Desfray%20-%20UMLWorkshop.pdf

[FOW03] Fowler M. “Model View Controller”, Patterns of Enterprise Application Architecture”. Addison-Wesley (2003), pp 330-332

[HES97] A.M. Hester; R.C.Borges; R. Ierusalimschy; “CGILua: A Multi-Paradigmatic Tool for Creating Dynamic WWW Pages”, Proceedings of the XI Brazilian Software Engineering Symposium (SBES’97) pp.347-360, Fortaleza, Brazil, 1997

http://www.tecgraf.pucrio.br/~anna/cgilua/cgilua.ps.gz

[ÎFM13a] IFML “The Interaction Flow Modeling Lenguaje”. (2013) http://www.ifml.org/?page_id=83

[IFM13b] IFML. “IFML by Example: Modeling an Online Bookstore”. 2013 ”http://www.ifml.org/wp-content/uploads/IFML-Bookstore-Example.pdf

[IFM13c] IFML. “Examples”. 2013. http://www.ifml.org/?page_id=99

[ISA95] Isakowitz, E. Stohr A. and Balasubramanian P. “RMM: a methodology for structured hypermedia design”. ACM (1995), 34-44.

[KLE03] Kleppe A., Warmer J., Bast W. “MDA explained: the model driven architecture: practice and promise”. Addison-Wesley Professional (2003)

[KOC08] Koch, Knapp, Zhang, Baumeister.Uml-Based Web Engineering, Chapter 7 “Web Engineering: Modelling and Implementing Web Applications”, Springer London (2008), pp 157-191

[LAM11] María Jesús Lamarca Lapuente.” Hipertexto: el nuevo concepto de documento en la cultura de la imagen”.

http://www.hipertexto.info/documentos/oohdm.htm

[LAMNN] María Jesús Lamarca Lapuente. “Modelo OOHDM: Hipertexto: El nuevo concepto de documento en la cultura de la imagen.”

http://www.hipertexto.info/documentos/oohdm.htm

[LER96] R. Lerusalimschy, L. H. de Figueiredo and W. Celes, "Lua – an extensible extension language", Software: Practice & Experience 26#6 (1996) 635-652.

http://www.tecgraf.puc-rio.br/lua.

[LIF12] LIFIA (Laboratorio de Investigación y Formación en Informática Aplicada). “Investigación”, Universidad Nacional de La Plata

http://www.lifia.info.unlp.edu.ar/lifia/es/

[OMG00] Object Mannagement Group. OMG XML Metadata Interchange (XMI) Specification - Version 1.0 - formal/00-06-01. (2000)

http://www.omg.org/cgi-bin/doc?formal/00-06-01.pdf

[OMG05a] Object Mannagement Group. Unified Modeling Language Specification - Version 1.4.2 - formal/05-04-01. (2005).
http://www.omg.org/cgi-bin/doc?formal/05-04-01.pdf

[OMG05b] Object Mannagement Group. MOF 2.0/XMI Mapping Specification - Version 2.1 - formal/05-09-01. (2005).
http://www.omg.org/cgi-bin/doc?formal/05-09-01.pdf

[OMG05c] Object Mannagement Group. OMG XML Metadata Interchange (XMI) Specification - Version 2.0.1 - formal/05-05-06. (2005)
http://www.omg.org/cgi-bin/doc?formal/05-05-06.pdf

[OMG13a] “OMG`s MetaObject Facitilty”(2013).
http://www.omg.org/mof

[OMG13b] Object Mannagement Group. OMG: MOF 2 XMI Mapping. Version 2.4.1, - formal/2013-06-03. (2013)
http:/www.omg.org/spec/XMI

[PIN08] Pineda C. “Un Método de Desarrollo de Hipermedia Dirigido por Modelos”. Tesis Doctoral. Universidad Politécnica de Valencia. (2008)
http://riunet.upv.es/bitstream/handle/10251/3884/tesisUPV2961.pdf

[PON97] Pontes, R.C.A., “An Environment to Support Hypermedia Applications in the WWW” , MSc thesis, PUC-Rio, 1997 (in Portuguese).

[RODNN] Rodríguez Fernández María, Valdez Gayo Juan Francisco: "CINEMEDIA ASTUR: Herramienta para la generación de Aplicaciones Hipermedia con soporte para OOHDM".

[ROS08] Rossi, Schwabe. & D. Olsina."Web Engineering: Modelling and implementing web applications, 2008.

[ROS96] Rossi, A. Garrido and S. Carvalho: "Design Patterns for Object-Oriented Hypermedia Applications". Pattern Languages of Programs 2, Vlissides, Coplien and Kerth eds., Addison Wesley, 1996.

[ROS97] G. Rossi, D. Schwabe and A. Garrido: "Design Reuse in Hypermedia Applications Development" Proceedings of ACM International Conference on Hypertext (Hypertext'97), Southampton, April 7-11, 1997, ACM Press.

[ROS99] G. Rossi, Daniel Schwabe, Fernando Lyardet: "Web Application Models Are More than Conceptual Models".

[SCH01] Daniel Schwabe, Gustavo Rossi, Luiselena Esmeraldo, Fernando Lyardet. "Web Design Frameworks: An Approach to Improve Reuse in Web Applications". 2001.

[SCH95] D. Schwabe and G. Rossi:, "The Object Oriented Hypermedia Design Model", Comm. of the ACM, Vol. 38, #8, pp45-46 Aug. 1995. (http://irss.njit.edu:5080/cgi-bin/bin/option.csh?sidebars/schwabe.html).

[SCH96a] Schwabe, G. Rossi and S. Barbosa: "Systematic Hypermedia Design with OOHDM". Proceedings of the ACM International Conference on Hypertext (Hypertext'96), Washington, March 1996.

[SCH96b] SCHWABE, D. ROSI, G.; BARBOSA, S." Systematic Hypermedia Application Design with OOHDM". Washington DC, March, 16-20, 1996.

[SCH98] Schwabe D. y Rossi G. "An object oriented approach to Web-based applications design". Theor. Pract. Object Syst. Volume 4, Issue 4 (1998), pp 207-225.

[SCH98a] Daniel Schwabe , Gustavo Rossi. "Developing Hypermedia Applications using OOHDM".

[SCH98b] Daniel Schwabe and Gustavo Rossi, "An Object Oriented Approach to Web-Based Application Design".

(http://www-di.inf.puc-rio.br/schwabe//papers/TAPOSRevised.pdf)

[SCH99] Daniel Schwabe, Rita de Almeida Pontes and Isabela Moura: “OOHDM-Web: An Environment for Implementation of Hypermedia Applications in the WWW”.

[SIL13] Silva Mateo. WEBINAR. “IFML” (2013). http://www.slideshare.net/silvamatteo/webinar-ifml-en-espaol

[SILNN] Darío Andres Silva, Bárbara Mercerat: “Construyendo aplicaciones web con una metodología de diseño orientado a objetos”.

[TRU07] Juan Trujillo, Emilio Soler, Eduardo Fernandez-Medina y Mario Piattini. “Un conjunto de transformaciones qvt para el “modelado de almacenes de datos seguros”. http://ceur-ws.org/Vol-227/paper05.pdf?iframe=true&width=90%&height=90%

[VER12] Vera Pablo, Pons Claudia, Giulianelli Daniel, Rodríguez Rocío. “Utilizando el Enfoque MDA para la Construcción de Aplicaciones Web Móviles Centradas en los Datos”. Workshop de Investigadores en Ciencias de la Computación. Misiones, Argentina. http://sedici.unlp.edu.ar/bitstream/handle/10915/18937/Documento_completo.pdf?sequence=1

[W3C08a] W3C, “Default Delivery Context” (2008).

http://www.w3.org/TR/mobile-bp/#ddc

[W3C08b] W3C, “Mobile Web Best Practices 1.0”, 2008

http://www.w3.org/TR/mobile-bp/

[W3C99] W3C, “XSL Transformations (XSLT)”. Version 1.0 (1999).

http://www.w3.org/TR/xs

[W3C10] W3C, “Mobile Web Application Best Practices”, 2010.

http://www.w3.org/TR/mwabp/

Páginas Visitadas.

http://msdn.microsoft.com/es-es/library/ms256069%28v=vs.80%29

http://developer.expressionz.in/blogs/es/tag/xslwhen/

http://lineadecodigo.com/categoria/xslt/

http://www.w3.org/TR/xslt20/

http://www.w3schools.com/xsl/default.asp

http://www.w3.org/TR/xslt

http://www.ibm.com/developerworks/library/x-xslt/

http://cs.au.dk/~amoeller/XML/transformation/

http://www.ebooksdownloadfree.com/Programming-general/XSLT-Programmer-s-Reference-BI19036.html

8. Anexos

8.1. Anexo A - QVT

Enfoque Hibrido

Un enfoque híbrido básicamente combina las características que proveen los lenguajes puramente declarativos (los cuales se limitan a escenarios donde los metamodelos de entrada y salida poseen una estructura similar y consecuentemente permiten realizar un mapeo) con la de los lenguajes puramente imperativos (manejan un mayor número de escenarios y poseen un nivel de abstracción menor).

QVT (Query/View/Transformation)

Estándar para la transformación de modelos, que depende de otros dos estándares (los cuales son OCL 2.0 y MOF 2.0) para la especificación de modelos y metamodelos. QVT solo soporta las transformaciones de modelo a modelo, donde un modelo es una entidad que se ajusta a un metamodelo.

La parte declarativa de QVT se separa en una arquitectura de dos capas en dos niveles de abstracción. Dichas capas son:

- **Relations:** lenguaje que soporta reconocimientos de patrones complejos, creación de objetos mediante templates y creación implícita mediante trazas entre elementos de los modelos origen/destino.
- **Core:** lenguaje definido usando extensiones mínimas de EMOF y OCL. Las trazas son explícitamente definidas como modelos MOF y sus instancias se crean y eliminan de igual forma que para otros objetos.

La parte imperativa está formada por dos componentes que extienden a los lenguajes Core y Relations:

- El lenguaje **Operational Mappings**, cuya sintaxis provee constructores que, comúnmente, se encuentran en lenguajes de programación imperativos.
- La implementación **Black-Box**, para invocar facilidades de transformaciones expresadas en otros lenguajes. Permite integrar bibliotecas y transformaciones no realizadas con QVT.

QVT Relations

Es un lenguaje relacional, que brinda una especificación declarativa de las relaciones entre los modelos MOF, soporta reconocimiento de patrones y puede ser representado

mediante una declaración textual o gráfica. Una transformación necesita declarar parámetros para manejar los modelos involucrados en la transformación.

Las transformaciones se describen a través de:

- La enumeración de los metamodelos participantes.
- Un conjunto de reglas que determinen las relaciones existentes entre los términos de los metamodelos.
- Un conjunto de dominios por regla que se ajusta al conjunto de términos para los que se expresan relaciones.
- Un conjunto de patrones que cumplen con la estructura de los términos y a los cuales se les aplican operaciones OCL.

Una relación puede incluir condiciones de aplicación o “precondiciones”, cláusulas When (cuando) y/o cláusulas Where (donde); pueden ser del tipo Top-Level o Non-Top-Level (comunes). Además existen definiciones KEY que se usan para poder definir claves que luego determinarán si las instancias de los objetos deben ser creadas o actualizadas durante la ejecución de la transformación.

En QVT Relations hay dos tipos de reglas, las cuales pueden ser de transformación o validación, las cuales quedan determinadas por el dominio de destino. Además, se debe tener en cuenta como se encuentra marcado el dominio, ya sea mediante la marca “CHECKONLY” (la cual comprueba si existe una correspondencia válida entre modelos) o la marca “ENFORCE” (fuerza a los términos a cumplir la relación).

El lenguaje otorga un mecanismo para organizar las reglas en módulos, siendo la organización de la regla orientada al objetivo (target-oriented), lo cual indica que existe una regla por cada tipo de elemento en el modelo destino. Las reglas se pueden “anidar” en función a la herencia existente en el modelo de origen.

Cuenta con soporte para Bidireccionalidad, ya que las reglas multidireccionales pueden ser especificadas. Al momento de la invocación, el usuario debe especificar la dirección de la ejecución de la transformación. Una transformación puede ser ejecutada en la dirección de un solo parámetro. El modelo especificado por este parámetro será el modelo destino de la transformación. En casos especiales, el modelo destino puede ser uno de los modelos de origen, por lo que el lenguaje soporta actualizaciones “in-place”.

También, posee la habilidad de generar actualizaciones en modelos de destino existentes que se basan en cambios realizados en modelos de origen, lo cual hace que QVT Relations provea incrementalidad. También cuenta con creación implícita de

declaraciones de clases para trazas y sus instancias para almacenar información durante la ejecución de la transformación.

- ***Transformaciones y tipos de modelos***

En el lenguaje Relations, una transformacion entre modelos se especifica como un conjunto de relaciones que se deben cumplir y mantener para que la transformación se exitosa. Un modelo candidato es cualquier modelo que se ajusta a un tipo de modelo, que es una especificación de que tipos de elementos de un modelo puedo tener. Los modelos candidatos se nombran y los tipos de elementos que pueden contener se limitan a aquellos que están dentro de un “paquete” de referencia.

Un ejemplo:

transformation umlRdbms (uml : SimpleUML, rdbms : SimpleRDBMS) {

En esta declaración nombrada “umlRdbms”, hay dos tipos de modelos candidatos, los cuales son “uml” y “rdbms”. El primer modelos declara el paquete “SimpleUML” como su metamodelo, mientras que “rdbms” declara como su metamodelo a “SimpleRDBMS” .

- ✓ ***Dirección de ejecución de una transformación***

Una transformacion invocada mediante un modo “ENFORCE” se ejecuta una dirección particular, seleccionando a uno de sus modelos candidatos como Objetivo. Este modelo objetivo, puede estar vacío, o puede contener elementos que se relacionan con la transformación. El procedimiento de ejecución de la transformación consiste, en primera instancia, chequear que es lo que se debe cumplir, para que, luego, en el caso en que falle la relación que se debe mantener (esto se debe al incumplimiento de requisitos), se cree, borre o modifique el modelo objetivo para hacer cumplir la relación.

- ***Relaciones y dominios***

Las relaciones en una transformación declaran restricciones que deben ser satisfechas por los elementos de los modelos candidatos. Una relación, definida por dos o más dominios y un par de cláusulas “WHEN” y “WHERE”, especifica la relación a mantener entre los elementos de los modelos candidatos.

Un dominio es una variable de tipo distinguido que se puede unir a un modelo de un determinado tipo. Tiene un patrón, que se puede ver como un gráfico de nodos de objetos, cuyas propiedades y enlaces de asociación se originan desde la instancia del

tipo de dominio. Un patrón puede ser visto como un conjunto de variables y de restricciones que los elementos del modelo con destino a dichas variables deben cumplir para que la unión sea válida.

Un ejemplo:

```
relation PackageToSchema
{
domain uml p:Package {name=pn}
domain rdbms s:Schema {name=pn}
}
```

En el ejemplo anterior, se declaran dos dominios que van a unir los elementos de los modelos “uml” y “rdbms”. Cada dominio especifica un patrón (un paquete con un nombre, y un esquema con un nombre), cuyas propiedades se unen a la misma variable de nombre “pn”, lo que implica que deben tener los mismos valores.

- ***Las clausulas When (cuando) y Where (donde)***

Una relación puede contener clausulas del tipo WHEN y WHERE, las cuales limitan a dicha relación. La clausula When, determina las condiciones que la relación debe mantener. En el ejemplo posterior, la relación ClassToTable se debe cumplir solo cuando la relación PackageToSchema se mantiene entre el paquete que contiene la clase y el esquema que contiene la tabla. Por otra parte, la cláusula WHERE determina las condiciones que deben satisfacer los elementos de los modelos que participan en la relación, y puede limitar a cualquiera de las variables en la relación y en sus dominios. En el ejemplo, en donde se cumple la relación ClassToTable, se debe de cumplir también la relación AttributeToColumn.

Ejemplo:

```
relation ClassToTable
{
domain uml c:Class {
namespace = p:Package {},
 kind='Persistent',
```

```
 name=cn
}
domain rdbms t:Table {
schema = s:Schema {},
name=cn,
column = cl:Column {
        name=cn+'_tid',
        type='NUMBER'},
            primaryKey = k:PrimaryKey {
            name=cn+'_pk',
            column=cl}
}

when {
         PackageToSchema(p, s);
        }

where {
         AttributeToColumn(c, t);
        }
 }
```

- ***Relaciones del tipo Top-Level***

Una transformación contiene dos tipos de relaciones, las cuales son del tipo Top-Level y Non-Top-Level. La ejecución de una transformación requiere que todas sus relaciones del tipo Top-Level se mantengan/cumplan, mientras que, por otro lado, las relaciones del tipo Non-Top-level se deben de cumplir/mantener solo cuando son invocadas directamente o de forma transitiva desde la clausula “WHERE” de otra relación.

Ejemplo:

```
transformation umlRdbms (uml : SimpleUML, rdbms : SimpleRDBMS) {
```

top relation PackageToSchema {…}

top relation ClassToTable {…}

relation AttributeToColumn {…}

}

La relación Top-Level tiene palabra clave “top” para distinguirlo sintácticamente. En el ejemplo, PackageToSchema y ClassToTable son relaciones Top-Level, mientras que AtribbuteToColumn es una relación del tipo Non-Top-Level.

- ***Modos Check y Enforce***

El dominio objetivo de una relación puede determinar si una relación puede ser forzada o no. Para ello debe ser marcado como un tipo “CHECKONLY” o “ENFORCED”. En el caso en el cual la transformación sea forzada en la dirección de un dominio marcado como “checkonly”, esta solamente es “chequeada” para ver si existen uniones válidas en el modelo pertinente que satisfagan la relación. En el caso en que una transformación se ejecute en la dirección de un modelo del tipo “enforce”, si falla el chequeo, el modelo destino se modifica para satisfacer la relación, lo cual implica una semantica del tipo check-before-enforce (chequear antes de forzar).

relation PackageToSchema

{

checkonly domain uml p:Package {name=pn}

enforce domain rdbms s:Schema {name=pn}

}

En el ejemplo anterior, el dominio del modelo “uml” se marcó como “checkonly”, mientras que el dominio de “rdbms” se marcó como “enforce”.

En el caso en que se ejecute en la dirección de uml y existiera un esquema en rdbms para el cual no se tenga un paquete correspondiente con el mismo nombre en uml, se reporta como una inconsistencia, por lo que el modelo es solamente chequeado. En el caso en que se ejecute la transformación en la dirección de rdbms, entonces, por cada paquete en el modelo uml, la relación chequea que existe un esquema con el mismo nombre en el modelo rdbms, y, en el caso de que este no exista, se crea un nuevo esquema en el modelo rdbms con el mismo nombre que el paquete contenido en uml. Por otra parte, si ejecutamos la transformación en dirección del modelo rdbms y no existe un paquete correspondiente con el mismo nombre en el modelo uml, el esquema contenido en el modelo rdbms será borrado.

Herramientas:

- Medini QVT: herramienta integrada en el entorno de desarrollo Eclipse, que permite la ejecución de transformaciones QVT expresadas con la sintaxis textual del lenguaje Relations.
- ModelMorf: Cuenta con soporte para realizar transformaciones in-place, operaciones BlackBox, trazas, herencia de transformaciones, sobrecarga de relaciones y ejecución incremental.
- Eclipse M2M: herramienta oficial de Eclipse open source que se encuentra en fase de desarrollo.

QVT CORE

- Brinda especificaciones declarativas de las relaciones entre los modelos MOF. Soporta reconocimiento de patrones sobre un conjunto de variables, y permite evaluar las condiciones de estas variables en un conjunto de modelos.
- Es igual de potente que QVT Relations y, a diferencia de este, su semántica se puede definir de una forma más simple. Por otro lado, a diferencia de QVT-R, el cual crea de forma implícita clases y objetos para mantener trazas y almacenar lo ocurrido en la transformación, en QVT-C las trazas deben ser definidas de forma explícita.
- Las transformaciones en QVT-C se define como un conjunto de “mappings” que declaran ciertas restricciones sobre los elementos de los modelos y el modelo de trazas. Una transformación se puede ejecutar en modo CHECKING (se encarga de validar las restricciones sobre los modelos involucrados y el modelo de trazas) o en modo ENFORCE (la transformación se ejecuta en una dirección particular determinada al seleccionar el modelo objetivo).
- Una transformación contiene mappings, el cual, a su vez, contiene uno o más dominios asociados al tipo de modelo que se define en una transformación. QVT-C permite hacer refinamiento de mappings, lo cual permite especializarlos en mappings más específicos (esto es similar al concepto de herencia de clases).
- También permite componer patrones. Un mapping especifica patrones para especificar los objetos a los cuales afectará. Un patrón es especificado por un

conjunto de variables, predicados y asignaciones, pudiendo depender de otros patrones.

- Como QVT-C y QVT-R cuentan con la misma 'expresividad', se puede traducir una transformación definida en lenguaje Relations a una equivalente al lenguaje Core con su respectiva semántica.

- ***Comparación con el lenguaje Relations***

El lenguaje Core soporta la unión de patrones sobre un grupo de variables, para lo cual evalúa esas variables contra un grupo de modelos. Su semántica se puede definir de una manera más simple, aunque las transformaciones que se describen utilizando a este lenguaje (Core) son mucho más extensas. Al lenguaje Core se lo puede implementar directamente, o se lo puede usar como referencia para la semánticas del lenguaje Relations, las cuales pueden ser asignadas al propio lenguaje Core mediante el uso de transformaciones entre lenguajes.

Crea de forma explícita trazas de clases y objetos para grabar que ocurrió durante la ejecución de la transformación.

- ***Transformación y tipos de modelos***

En el lenguaje Core, una transformación se especifica como un conjunto de asignaciones que declaran restricciones que se deben mantener entre los elementos candidatos y las trazas de modelos. Los modelos candidatos son nombrados, y los tipos de elementos que contienen se restringen por un tipo de modelo.

Una transformación se puede ejecutar de dos modos: el modo de control ("Cheking mode") y el modo forzado ("Enforcement mode"). En el modo de control, la ejecución de la transformación se verifica si las restricciones entre los modelos candidatos y las trazas de modelos se mantienen, y en el caso en que no, se hace un reporte de error. En el modo forzado, la ejecución de una transformación es en una dirección particular, la cual se define mediante la selección de uno de los modelos candidatos como modelo objetivos. La ejecución de la transformación verifica primero si se cumplen las restricciones, y luego, para modificar todas las restricciones que no se cumplen, modifica el modelo objetivo y el modelos de trazas.

- ***Asignaciones***

Una transformación contiene asignaciones, las cuales pueden tener varios dominios (hay casos en que no poseen ninguno). Estos dominios tienen un tipo de modelo asociado de la transformación. Además, se los identifica mediante la asignación y el tipo de modelo asociado, ya que no poseen nombres. En la ejecución de una transformación, cada

dominio selecciona uno de los modelos candidatos que es de interés para realizar la asignación.

Una asignación define una relación “uno a uno” entre los patrones de esa asignación. Esto significa que por cada unión exitosa de uno de los patrones debe existir una unión exitosa de otro patrón. La restricción “uno a uno” entre los patrones es controlada o ejecutada si existe una unión exitosa para cada patrón de vigilancia de la asignación

Cuando una transformación se ejecuta en modo de control, se ejecutan todas las asignaciones de la transformación, para lo cual se unen patrones, controlando así las restricciones “uno a uno”.

Cuando una transformación se ejecuta en el modo forzado en dirección de un modelo objetivo, cada asignación se ejecuta para aplicar la restricción “uno a uno”. Primero se unen los patrones, luego se ejecuta la restricción “uno a uno” (en el caso de que no se cumpla). El modo forzado solo realizará cambios a los elementos de la traza de modelo y del modelo objetivo asociado con el dominio y su tipo de modelo.

- ***Patrones***

Un patrón se define como un conjunto de variables, predicados y asignaciones. Los patrones pueden ser controlados y forzados. La unión de patrones puede resultar en obligaciones de valor de las variables, y forzar un patrón puede resultar en el cambio de modelos causando una nueva obligación de valor para las variables.

Los patrones pueden depender entre sí. Un patrón que depende de otro puede usar las variables de ese otro patrón en sus propios predicados y asignaciones, y se une usando obligaciones de valores de las variables producidas por la unión con ese otro patrón.

- ***Obligaciones***

La unión de patrones da como resultado varias obligaciones, incluso cero. Una obligación es un único grupo de todas las variables del patrón. Una obligación válida de patrones es un obligación donde todas las variables del patrón están unidas a un valor que no se “indefinido” (undefined), y donde todos los predicados del patrón evaluado son verdaderos.

- ***Dependencias de obligaciones***

Un patrón puede depender de otro patrón. Los predicados de un patrón que dependen de otro patrón pueden relacionar las variables declaradas en ese otro patrón. Unir un patrón que depende de otro involucra el uso de una obligación válida de ese patrón. Esto asegura que las variables tiene un valor cuando evalúan predicados.

- ***Modo de Control***

Las asignaciones pueden ser controladas, ya sea como parte de la ejecución de una transformación en modo de control o en el primer paso de la ejecución de una transformación en modo forzado. La ejecución de una transformación en modo de control producirá un error por cada incumplimiento de las restricciones de las asignaciones. Loa ejecución de una transformación en modo forzado obligará a reparar por cada incumplimiento de las restricciones de la asignación.

- ***Modo forzado***

En el tiempo de ejecución, un tipo de modelo de la transformación puede ser elegido como dirección de ejecución. La dirección de ejecución designa un grupo de dominios que asociados al tipo de modelo objetivo.

El modelo objetivo y el modelo de trazas pueden cambiar para cumplir con las obligaciones de las asignaciones. Los modelos solo cambian cuando las restricciones de las asignaciones no se cumplen. Los cambios pueden llevar a la creación de nuevos enlaces válidos o a borrar a los enlaces ya existentes de los patrones objetivos y de las trazas de patrones, para así poder hacer cumplir las restricciones de la asignación.

I. Las asignaciones pueden ser “default” o no. Las primeras solo asignan valores para satisfacer las restricciones de los “mapeos” (o asignaciones), por lo que no tienen un rol en la parte de control. Las asignaciones no “default” tienen un rol durante el control, donde el operador de asignación se reemplaza por el operador de igualdad.
II. Las variables pueden del tipo “realized” o no. Las primeras se caracterizan porque las instancias de este tipo de variables pueden ser creadas o borradas y por qué se usan en los modos de enlace y forzado, mientras que las segundas solo se usan para enlazar/unir valores durante el enlace.
III. Los dominios pueden ser forzados o no. Las asignaciones y las variables “realized” pueden ser definidas en los patrones de dominio forzados. Los patrones de dominios no forzados, por lo que se aplican solo las semánticas de control.

Herramientas:

- OptimalJ: Herramienta desarrollada por Compuware en el año 2001 para el lenguaje Java y fue descontinuada en el año 2008. Algunas de sus características principales son: desarrollo orientado a modelos, proceso orientado al desarrollo, transformación de patrones, diseñador de interfaz de usuario y desarrollo integrado.

OPERATIONAL MAPPING

Lenguaje estándar que proporciona implementaciones imperativas. Proporciona extensiones OCL con efectos laterales que dan un mayor estilo de procedimiento y una sintaxis más cercana a la programación imperativa.

Define operaciones de mappings que pueden ser usadas para implementar una o más relaciones a partir de una especificación “Relations”, cuando sea difícil proporcionar una especificación declarativa más pura. Estas operaciones pueden invocar otras operaciones de mappings con el objetivo de crear modelos de trazabilidad entre elementos de modelos.

Es posible escribir transformaciones completas utilizando operaciones de mapeo, las cuales se denominan “transformaciones operacionales”.

- Las transformaciones en QVT-O se expresan unidireccionalmente. Se codifica expresando en la transformación cuáles son los modelos de origen y objetivo explícitamente (parámetros) y se define un punto de entrada para la ejecución (main) que representa el código inicial al ser ejecutado para realizar la transformación.
- En QVT-O se pueden definir mapeos entre objetos del modelo de origen y del modelo destino, mediante Mapping Operations. También es posibles restringir estos mapeos mediante precondiciones (clausulas WHEN) y postcondiciones (clausulas WHERE).
- También, el lenguaje QVT-O provee tres mecanismos de reúso de reglas: herencia, merge y disyunción. Una regla puede heredar de otra regla (herencia), puede incluir una lista de reglas que la complementan (merge) o puede seleccionar una regla de un conjunto de reglas disjuntas (disyunción).
- Los dominios de una regla se especifican por una dirección que restringe los posibles cambios. Estos se pueden declarar como de entrada, salida o como de entrada/salida.
- QVT-O permite también definir “helpers”, los cuales son operaciones sobre elementos del modelo de origen que usualmente se utilizan para realizar consultas.
- Provee un mecanismo de bibliotecas para expresar definiciones que pueden ser reusadas en otras transformaciones. Las bibliotecas pueden definir tipos específicos y operaciones.
- QVT-O no brinda soporte para propagar cambios, por lo que los modelos de salida creados con anterioridad no pueden ser actualizados. Si bien no soporta incrementalidad, si permite la ejecución de transformaciones “in-place” y cuenta con

la habilidad de invocar operaciones de alto nivel en modelos. También cuenta con creación implícita de declaraciones de clases para trazas y de sus instancias.

El lenguaje Operational Mappings permite la definición de transformaciones usando un enfoque imperativo (transformaciones operacionales) o permite complementar transformaciones relacionales con operaciones imperativas utilizando las relaciones (enfoque hibrido).

- ***Transformaciones operacionales***

  Una transformación operacional representa una definición de una transformación unidireccional que se expresa imperativamente. Define una "firma" que indica que modelos se involucran en la transformación y define una operación de entrada para su ejecución (llamada "MAIN"). Similar a una clase, una transformación operacional es una entidad instanciable con propiedades y operaciones.

  Ejemplo:

```
transformation Uml2Rdbms(in uml:UML,out rdbms:RDBMS) {   /* punto de
                                                          entrada para la
                                                          ejecución de la
                                                          transformación */


main() {
 uml.objectsOfType(Package)->map packageToSchema();
        }
….
 }
```

  En el ejemplo anterior, se muestran la firma y el punto de entrada de una transformación llamada Uml2Rdbms, que transforma los diagramas de clase UML en tablas RDBMS.

  La firma de esta transformación declara que un modelo rdbms del tipo RDBMS será producido desde un modelo uml del tipo UML.

  En el ejemplo, la operación de entrada "MAIN", en primera instancia, recupera la lista de objetos de tipo "Package" (paquete) y luego les aplica una operación (packageToSchema) en cada paquete de la lista.

- ***Tipos de Modelos***

Los símbolos UML y RDBMS representan tipos de modelos. Un tipo de modelo se define por un metamodelos (paquetes MOF), tipo de conformidad (strick o effective), y un conjunto de condiciones opcionales. Los tipos de modelos introducen flexibilidad y precisión para escribir definiciones de transformaciones que son aplicables a metamodelos similares.

En el ejemplo Uml2Rdbms, los tipos de modelos UML y RDBMS se refieren a metamodelos implícitos. Cuando se usa la sintaxis textual, el escritor de transformación no está obligado a indicar que paquetes MOF se usan. Sin embargo, para construir la serialización XMI correspondiente, es obligatorio enlazar símbolo del tipo de modelo a una definición de Metamodelo existente.

Alternativamente a este enfoque, el escritor de la transformación indica que metamodelos están siendo usados. Esto es hecho por una declaración explícita que se ubica después de la firma de la transformación.

modeltype UML uses SimpleUml ("http://omg.qvt-examples.SimpleUml");

modeltype RDBMS "strict" uses SimpleRdbms;

La primera declaración determina que el modelo de tipo UML se define mediante el metamodelo SimpleUml y asume un tipo de conformidad del tipo “effective”, es cuál es el default. Esto implica que la representación XMI de la definición de la transformación debe ser construida usando la definición del metamodelo SimpleUml. Sin embargo, la conformidad “effective” permite el pasaje de modelos que no son necesariamente “instancias” de este metamodelo. El modelo de entrada puede ser chequeado para determinar si es una entrada válida para la transformación, comparando con la estructura de los elemento del modelo definido por el metamodelo SimpleUml.

Cuando se declara explícitamente un tipo de modelo, puede darse una absoluta URI para el metamodelo de referencia. Además, la palabra clave “where” se usa para definir las restricciones adicionales del tipo de modelo.

Ejemplo:

modeltype UML uses SimpleUml("http://omg.qvt-examples.SimpleUml")

where { self.objectsOfType(Class)->size()>=1};

En el ejemplo anterior se da una nueva definición del tipo de modelo “UML”, la cual impone al modelo a contener al menos a una clase.

- ***Bibliotecas***

Una biblioteca contiene definiciones que pueden ser usadas por las transformaciones. Deben definir tipos específicos y operaciones (como pedir operaciones o constructores de metaclases). Una biblioteca es “importada” a través de una o dos comodidades de importación disponibles: “Access” o “extensión”. La biblioteca estándar de QVT, llamada Stdlib, es una biblioteca desde la cual se predefine que no necesita ser importada explícitamente dentro de las definiciones de transformación.

La siguiente declaración define una biblioteca llamada MyUmlFacilities, la cual define una lista de operaciones de pedido en los modelos UML,

```
library MyUmlFacilities(UML);

query UML::Class::getAllBaseClasses() : Set(Class);

query UML::Element::getUsedStereotypeNames() : Set(String);
```

La siguiente declaración muestra el uso de la biblioteca en la transformación UmlCleaning .

```
transformation UmlCleaning(inout umlmodel:UML14)

extends MyUmlFacilities(UML14),

access MathUtils;

var allSuper : Set(Class); // variable global

main () { allSuper := umlmodel.objectsOfType(Class)-
>collect(i|i.getAllBaseClasses());

// ….

        }
```

La transformación UmlCleaning extiende la biblioteca MyUmlFacilities, indicando que todas las operaciones definidas en esta biblioteca se comportan como su hubieran sido definidas por la transformación UmlCleaning. El tipo de modelo “UML14” es el nombre de un símbolo local.

- ***Operaciones de mapeo (Mapping operations)***

Una operación de mapeo es una operación que implementa mapeos entre una o más elementos del modelo fuente en uno o más elementos del modelo destino.

Sintácticamente se describe mediante una firma, una “custodia” (clausula “When”), un cuerpo de mapeo, y una pos condición (clausula “Where”). Incluso si no está notada explícitamente en la sintaxis correcta, una operación de mapeo siempre es una refinación de una relación implícita que contiene a las clausulas “When” y “Where”.

La operación de mapeo packageToSchema siguiente define como un paquete UML debe ser transformado en un esquema RDBMS.

mapping Package::packageToSchema() : Schema

when { self.name.startingWith() <> "_"}

{

name := *self*.name;

table := *self*.ownedElement->map class2table();

}

La relación implícita asociada a esta operación de mapeo tiene la siguiente estructura:

relation REL_PackageToSchema {

checkonly domain:uml (self:Package)[]

enforce domain:rdbms (result:Schema)[]

when { self.name.startingWith() <> "_" }

}

La operación de mapeo packageToSchema se comporta como cualquier operación que fuese definida en el paquete de la metaclase UML. La variable “self” se refiere a una instancia del paquete de metaclase siendo pasado a la operación. La clausula When restringe la ejecución del cuerpo. En el ejemplo, si se pasa un paquete cuyo nombre empieza subrayado, la operación simplemente regresa un valor nulo (Null), por el contrario, el cuerpo es ejecutado.

La sintaxis general del cuerpo de una operación de mapeo es la siguiente:

mapping <dirkind0> X::mappingname

(<dirkind1> p1:P1, <dirkind2> p2:P2) : r1:R1, r2:R2

when { … }

where { … }

{

init { … }

population { … }

end { … }

}

Donde <dirkindN> hace referencia a los tipos de direcciones in/inout/out (entrada, entrada-salida, salida).

La sección de “init” contiene el código a ser ejecutado después de la instanciación de las salidas declaradas. La sección de “population” contiene el código que se encargará de “poblar” los parámetros de resultado, y la sección de “end” contiene el código adicional que se ejecutará antes de salir de la operación.

Entre las secciones de inicialización y población (“initialization” y “populate”), hay una sección implícita, llamada sección de instanciación (“instantiation section”), la cual crea todos los parámetros de salida que tienen el valor nulo (“NULL”) al final de la sección de inicialización. Esto quiere decir que, con el fin de devolver un objeto existente en lugar de crear uno nuevo, simplemente se le asigna el parámetro de resultado dentro de la sección de inicialización.

- ***Creación de objetos y población***

El lenguaje operational mappings define una facilidad específica de Alto nivel que permite crear y/o actualizar modelos de elementos:

La construcción “expresión de objeto” (object expression), para la cual se usa la palabra clave “object”.

object s:Schema {

name := self.name;

table := self.ownedElement->map class2table();

}

En el ejemple anterior, la expresión de objeto se refiere a una variable llamada “s”, que necesariamente debe ser del tipo de esquema.

- ***Composición de transformaciones***

  La composición de transformaciones es una característica esencial para las composiciones largas y complejas. Para este fin, el lenguaje permite instanciar e invocar transformaciones explícitas.

```
transformation CompleteUml2Rdbms(in uml:UML,out rdbms:RDBMS)
        access transformation UmlCleaning(inout UML),
        extends transformation Uml2Rdbms(in UML,out RDBMS);72
        main() {
                var tmp: UML = uml.copy();
                var retcode := (new UmlCleaning(tmp))->transform(); // performs
                the "cleaning"
                if (not retcode.failed())
                        uml.objectsOfType(Package)->map packageToSchema()
                        else raise "UmlModelTransformationFailed";
                }
```

  En el ejemplo se puede observar el uso de los mecanismos “Access” y “Extension”. Acces se comporta como un tradicional paquete de importación, mientras que la semántica de “extension” combina la importación de paquetes y del paradigma de herencia de clases.

Herramientas

- Borland Togheter: Utiliza QVT-O como lenguaje para realizar transformaciones de modelos y cuenta con soporte para sintaxis resaltada, validación y autocompletado de código OCL 2.0. Cuenta además con otras funcionalidades como la generación de documentación, herramientas para la creación de DSLs, modelado de procesos de negocio, de datos y diagramas UML. Provee de soporte a usuarios mediante ejemplos, un tutorial y lista de FAQs.
- Smart QVT: Se distribuye como un plugin para Eclipse que ejecuta sobre EMF y cuenta con licencia EPL. Provee soporte a usuarios por medio de lista de correos, grupo de noticias, foro de discusión y manual.
- Eclipse M2M Operational QVT: Entre sus principales características se encuentran: el resaltado de código, el marcado de errores, la asistencia para autocompletado de

código y la existencia de un debugger. Provee soporte a usuarios por medio de grupo de noticias, foro de discusión, ejemplos y documentación.

IMPLEMENTACIÓN BLACK-BOX

Es un mecanismo que permite enlazar códigos escritos en distintos lenguajes.

Para esto, se utiliza el lenguaje Relations para derivar una operación MOF de una transformación. Esta operación será el punto de enlace con el lenguaje deseado, que deberá definir la operación a ejecutar con la misma signatura, y soportar un enlace con la implementación MOF que se utilice.

Las operaciones MOF deben poder derivarse de Relations haciendo posible integrarlas con cualquier implementación de una operación MOF con la misma signatura. Esto resulta beneficioso por las siguientes razones:

- Permite codificar algoritmos complejos en cualquier lenguaje de programación con correspondencia a MOF.
- Permite el uso de bibliotecas específicas de dominio para calcular ciertas propiedades del modelo.
- Permite ocultar la implementación de algunas partes de una transformación.

Pero, esta integración puede resultar peligrosa ya que la implementación asociada a ella tiene libre acceso a las referencias de objetos en modelos. Las implementaciones Black-Box no tienen una relación implícita hacia Relations, y cada Black-Box debe implementar explícitamente una relación que sea responsable de conservar la trazabilidad entre los elementos del modelo relacionado con la implementación de la operación MOF.

Artículos relacionados

"Transformación de modelos aplicada a la definición genérica de casos de uso utilizando QVT y RTG (Reglas de transformación de grafos). [DAN11]"

Vinculación:

El artículo describe un método que permite mejorar el proceso de desarrollo de software en el cual se hace uso de QVT (Query/View/Transformation) para realizar las transformaciones entre modelos.

Resumen:

El objetivo del trabajo es realizar un aporte a la mejora del proceso de desarrollo de software. En este contexto, se definen Plantillas Genéricas con el fin de otorgar una aplicación práctica y estandarizada a procesos de desarrollos basados en caso de

uso, permitiendo, también, plantear soluciones genéricas y estandarizadas en etapas como las de captura de requisitos, análisis y diseño. Se presenta un metamodelo para definir plantillas genéricas que se utilizan en la etapa de captura de requisitos y se propone realizar transformaciones de los modelos instanciados a este nuevo metamodelo mediante la aplicación de QVT y Reglas de Transformación de Grafos. El uso de las ya mencionadas plantillas se debe a que estas permiten plantear soluciones genéricas y estandarizadas en diversas etapas, plantear soluciones a problemas que requieren igual tratamiento que otros ya planteados, analizadas y resueltos, y, además, permite reducir la cantidad de documentación producida, lo cual facilita tanto la lectura como la comprensión de cada uno de los casos de uso, como así también reducir la dificultad que se genera para comprender diversos modelos de solución planteados para resolver el mismo tipo de problemas.

Con el fin de lograr el objetivo propuesto, se optó por utilizar el concepto de MDA (Arquitectura Dirigida por Modelos, o “Model Driven Architecture”), el cual propone cuatro niveles de abstracción que componen la arquitectura por modelos: CIM (Modelo independiente de computo), PIM (modelo independiente de plataforma), PSM (modelo especifico de plataforma) y la aplicación final. Algunos de los niveles mencionados anteriormente que están estrechamente relacionado con el proyecto, es el PIM, ya que este modela la funcionalidad y la estructura de un sistema de información sin tener en cuenta los “detalles tecnológicos” que tendrá la plataforma en la cual se utilizara el sistema, y el PSM, el cual se encarga de definir los modelos específicos de la plataforma en la que se ejecutará el sistema.

En este trabajo se plantea la utilización de QVT para definir transformaciones entre modelos, las cuales describen relaciones entre un metamodelo fuente F y un metamodelo objetivo O, ambos especificados en MOF (Meta Object Facility). La transformación definida se utiliza para obtener un modelo objetivo (instancia del metamodelo O), a partir de un modelo fuente (instancia del metamodelo F).

Respecto al desarrollo del proyecto, se utilizan los modelos propuestos por MDA para el desarrollo de software. Una vez definidos estos modelos se definen las transformaciones que generan nuevos modelos, lo cual permite ir de modelos más abstractos a otros más concretos. En una primera etapa, se trabaja en la formalización de las Plantillas Genéricas para describir los casos de uso a través de un metamodelo UML. Luego se transforman los modelos pre-establecidos en las plantillas genéricas, utilizando QVT y las reglas de transformación de grafos para generar los modelos que corresponden a las etapas de análisis y diseño. Esto permite

realizar el acercamiento a la automatización de la construcción del software aplicada a aquellos procesos de desarrollo dirigidos por casos de uso.

"Un conjunto de transformaciones qvt para el Modelado de almacenes de datos seguros. [tru07]"

Vinculación:

El artículo se relaciona con el concepto de QVT, ya que utiliza dicho concepto para realizar una seria de transformaciones con el fin de poder modelar una serie de almacenes de datos seguros.

Resumen:

El artículo propone un método el cual utiliza QVT para poder transformar al esquema lógico relacional de un Almacén de datos (Data Warehouse o DW) toda la información de seguridad representada en el modelo conceptual multidimensional. Para lograr dicho fin, se utiliza el concepto de arquitectura dirigida por modelos (MDA), más precisamente se utiliza una propuestas del mismo, el cual es QVT (Query/View/Transformation). MDA parte de la idea de separar la especificación de la operación de un sistema de los detalles de su plataforma. De aquí surge la especificación de un Modelo Independiente de la Plataforma (PIM), el cual podrá ser transformado en uno o varios Modelos Específicos de Plataforma (PSM), con el fin de incluir la información acerca de la tecnología que será usada en una plataforma específica. Para realizar la Transformación de PIM a PSM, se considera la aproximación declarativa de QVT, la cual ofrece la notación diagramática y textual para definir la transformación. Una transformación tiene Dos o más dominios, un dominio de relación (checkonly o enforced), clausulas Where y When, y una transformación con dos tipos de nivel (Top Level y Non-Top-Level).

Las reglas de seguridad y auditoria especificadas a nivel conceptual en el modelado multidimensional de los almacenes de datos no pueden ser representadas directamente en el modelo relacional, por lo cual se tiene un "HUECO SEMÁNTICO" entre los esquemas conceptual y lógico. Para cubrir dicho "hueco", se utiliza MDA, mediante las transformaciones QVT. El uso de QVT permite transformar los modelos conceptuales multidimensional seguro en un esquema lógico relacional seguro. Esto, sumado a las definiciones de un PIM y un PSM seguros, permite definir una arquitectura MDA multidimensional segura. La principal ventaja de esta arquitectura se presenta en un ahorro de tiempo y esfuerzo para los diseñadores.

8.2. Anexo B - XSLT.

✓ **Generalidades**

XSLT (Transformación de procesos basada en template).

Basado en reglas de Template
template rule = template pattern + template body.
<xsl:template match="pattern"> body </xsl:template>The Transformation Process (Transformación de procesos basada en template)

✓ **Tipos de Datos en XSLT**

Existen 5 tipos disponibles:

1. Boolean
2. Number
3. String
4. Node-set
5. External object

✓ **Elementos**

Elementos	Detalles
xsl:apply-imports	Invoca una regla de plantilla reemplazada.
xsl:apply-templates	Dirige el procesador XSLT para que busque la plantilla adecuada que se debe aplicar, según el tipo y el contexto del nodo seleccionado.
xsl:attribute	Crea un nodo de atributo y lo asocia a un elemento resultante.
xsl:attribute-set	Define un conjunto de atributos con nombre.
xsl:call-template	Invoca una plantilla por nombre.
xsl:choose	Proporciona múltiples pruebas condicionales junto con los elementos <xsl otherwise> y <xsl when>.
xsl:comment	Genera un comentario en el resultado.
xsl:copy	Copia el nodo actual del origen al resultado.
xsl:copy-of	Inserta subárboles y fragmentos del árbol de resultados en el árbol de resultados.
xsl:decimal-format	Declara un formato-digital, que controla la interpretación de un modelo de formato utilizado por la función format-number.
xsl:element	Crea un elemento con el nombre especificado en el resultado.
xsl:fallback	Llama al contenido de la plantilla que puede proporcionar un sustituto razonable al comportamiento del nuevo elemento cuando se encuentre.
xsl:for-each	Aplica una plantilla repetidamente, aplicándola a su vez en cada nodo de un conjunto.
xsl:if	Permite obtener fragmentos de plantillas condicionales simples.
xsl:import	Importa otro archivo XSLT.
xsl:include	Incluye otro archivo XSLT.

Elementos	Detalles
xsl:key	Declara una clave para utilizar con la función key() en expresiones de lenguaje de ruta XML (XPath).
xsl:message	Envía un mensaje de texto al búfer del mensaje o al cuadro de diálogo del mensaje.
xsl:namespace-alias	Sustituye el prefijo relacionado con un espacio de nombres dado por otro prefijo.
xsl:number	Inserta un número con formato en el árbol de resultados.
xsl:otherwise	choose> y <xsl when>.
xsl:output	Especifica las opciones que se deben utilizar a la hora de serializar el árbol de resultados.
xsl:param	Declara un parámetro con nombre que se puede utilizar dentro de un elemento <xsl stylesheet> o un elemento <xsl template>. Permite especificar un valor predeterminado
xsl:preserve-space	Conserva los espacios en blanco en un documento.
xsl:processing-instruction	Genera una instrucción de proceso en el resultado.
msxsl:script*	Define variables y funciones globales para extensiones de secuencias de comandos.
xsl:sort	Especifica los criterios de ordenación para las listas de nodos seleccionadas por <xsl for-each> o <xsl apply-templates>.
xsl:strip-space	Elimina espacios en blanco en un documento.
xsl:stylesheet	Especifica el elemento de documento en un archivo XSLT. El elemento de documento contiene otros elementos XSLT.
xsl:template	Define una plantilla reutilizable para generar los resultados deseados para los nodos de un tipo y contexto en particular.
xsl:text	Genera texto en el resultado.
xsl:transform	Lleva a cabo la misma función que <xsl stylesheet>.
xsl:value-of	Inserta el valor del nodo seleccionado como texto.
xsl:variable	Especifica un valor enlazado de una expresión.
xsl:when	Proporciona múltiples pruebas condicionales junto con los elementos <xsl choose> y <xsl otherwise>.
xsl:with-param	Pasa un parámetro a una plantilla.

✓ **Funciones**

Elementos	Detalles
current	Devuelve un conjunto de nodos que tiene el nodo actual como único miembro.
document	Proporciona un modo de recuperar otros recursos XML desde la hoja de estilos XSLT más allá del dato inicial proporcionado por la secuencia de entrada.
element-available	Devuelve un valor verdadero únicamente si el nombre expandido es el nombre de una instrucción.
format-number	Convierte el primer argumento en una cadena mediante la cadena de modelo de formato que especifica el segundo argumento.
function-available	Devuelve un valor verdadero si la función se encuentra en la biblioteca de funciones.
generate-id	Devuelve una cadena que sólo identifica el nodo del argumento node-set que está en primer lugar en el orden del documento.
key	Recupera elementos previamente marcados con una instrucción <xsl
key>.	
node-set	Convierte un árbol en un conjunto de nodos. El nodo resultante siempre contiene un único nodo, que es el nodo raíz del árbol.
system-property	Devuelve un objeto que representa el valor de la propiedad del sistema que identifica el nombre.
unparsed-entity-uri	Devuelve declaraciones de entidades sin analizar de la definición del tipo de documento (DTD) del documento de origen.

✓ **Filtro:**

Elementos	Detalles
xsl:attribute	Crea un nodo de atributo y lo asocia a un elemento resultante. <xsl:attribute name = "attribute-name" namespace = "uri-reference"> </xsl:attribute>
xsl:attribute-set	Define un conjunto de atributos con nombre. <xsl:attribute-set name = QName use-attribute-sets = QNames> atributo1 atributo2 atributo3 … </xsl:attribute-set>
xsl:call-template	Invoca una plantilla por nombre. <xsl:call-template

Elementos	Detalles
	name = QName </xsl:call-template>
xsl:choose xsl:otherwise y xsl:when	Proporciona múltiples pruebas condicionales junto con los elementos. Choose <xsl:choose> </xsl:choose> Otherwise <xsl:otherwise> </xsl:otherwise> When <xsl:when test = boolean-expression </xsl:when>
xsl:for-each	Aplica una plantilla repetidamente, aplicándola a su vez en cada nodo de un conjunto. <xsl:for-each select = Expression </xsl:for-each>
xsl:if	Permite obtener fragmentos de plantillas condicionales simples. 1) NO TIENEN ELSE 2) NO ADMITE “a<b” (menor) 3) Solo admite : =, !=, >=, > y not( ) 4) Para comparar texto utilizae comillas simples [ej: <xsl:if test="title='Greatest Hits'">] <xsl:if test = boolean-expression </xsl:if>
xsl:message	Envía un mensaje de texto al búfer del mensaje o al cuadro de diálogo del mensaje. Si el valor de “terminate” es “yes” se corta la transformación en esta instrucción mostrando el mensaje de error <xsl:message terminate = "yes" | "no" > </xsl:message>
xsl:param	Declara un parámetro con nombre que se puede utilizar dentro de un elemento <xsl:stylesheet> o un elemento <xsl:template>. Permite especificar un valor predeterminado. <xsl:param name = QName select = Expression </xsl:param>
xsl:sort	Especifica los criterios de ordenación para las listas de nodos seleccionadas por <xsl:for-each> o <xsl:apply-templates>. USAR LOS SORT DENTRO DE LAS ITERACIONES <xsl:sort select = string-expression lang = { nmtoken } data-type = { "text" | "number" | QName } order = { "ascending" | "descending" } case-order = { "upper-first" | "lower-first" }

Elementos	Detalles
	/>
xsl:template	Define una plantilla reutilizable para generar los resultados deseados para los nodos de un tipo y contexto en particular. <xsl:template name= Qname match = Pattern priority = number mode = QName </xsl:template>
xsl:variable	Especifica un valor enlazado de una expresión. <xsl:variable name = QName select = Expression </xsl:variable>
xsl:with-param	Pasa un parámetro a una plantilla. <xsl:with-param name = QName select = Expression </xsl:with-param>

✓ **Particularidades**

Este lenguaje tiene algunas particularidades en cuanto a su escritura que son importantes:

- No dejar espacio en blanco entre los <> y las estructuras internas
  Ej1: ~~< xsl:if >~~
  Ej2: ~~<xsl: if>~~
- Espacio en blanco:

- Imprimir texto o valores:
  Texto
- Agregar estilo a lo impreso :
  Texto con formato
- Tiene un error ENORME al hacer cálculos números con distinta cantidad de decimales o terminan en decimal 0:
  - Ejemplo1:
    10.90*4.52=49.268(real)≠ 49.267999999999994(compilador)
  - Ejemplo2 [7decimales x 7decimales]:
    10.9994562*4.5225687=49.74579632714094(real)=49.74579632714094(compilador)
  - Formato de muestra de variables numéricas:
    - Funcion de formato numérico: **“format-number($variable,’formato’)”**
    - Cantidad de decimales:
      - **<xsl:variable** name="variable" select="'7.35001'" **/>**

- **<xsl:value-of** select="format-number($variable, '#.00')" **/>**
- *<!-- El resultado es: 7.35 -->*

- Para **dar formato**, podemos utilizar los siguientes caracteres:
  - **'0'**: Los ceros, indican dígitos obligatorios, ignorando los ceros a la izquierda o derecha si son en la parte entera o decimal. Si por ejemplo tenemos el 34 y le aplicamos como formato 0000, nos quedaría tal que 0034. Lo mismo ocurre con los decimales, si tenemos nuevamente 34, el resultado de aplicar 0000.00 seria 0034.00.
  - **'#'**: La almohadilla (#), sirve para dar formato a los dígitos ignorando los ceros innecesarios. Por ejemplo, si tenemos el 0034 y el 0034.0500 y le aplicamos el formato #.#, nos dará como resultado 34 en primer lugar y 34.1 en segundo lugar. OJO, que redondea los números, si pusiéramos al 0034.0500 dos almohadillas tal que #.##, quedaría 34.05. Tener en cuenta, además de que redondea los números, que ignorara todos los ceros que haya después de la posición que ocuparía la almohadilla, a diferencia del '0', que mantendría esos ceros.
  - **'.'**: El punto se utiliza para establecer el limitador decimal, para diferenciar entre la parte entera y la decimal.
  - **','**: La coma se utiliza para indicar si queremos y donde queremos situar los separadores de miles, por ejemplo, si tenemos 1000 y le aplicamos un formato tal que # o ####, el resultado siempre sera 1000, pero si aplicamos el formato #,### nos quedara 1,000, que manteniendo el formato y pasando 12350, quedaría 12,350 como resultado.
  - **'%'**: Devuelve el resultado en %, como pasa en otros programas como el Excel, cuando le aplicamos el %, nos multiplicara por 100 y mantendrá el símbolo de porcentaje al final, tal que 7 con formato #% quedaría 700% como resultado.
- No se puede cambiar el valor de una variable
- **NO RESTA DOS VARIABLES:**
  - Para restar hacer $a+(-1)*$b que, extrañamente SI funciona

8.3. Anexo C.

- ✓ **Clase Interoperabilidad.**

```
using Common.DBEntitties;
using Data;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using System.Xml;
using System.Xml.Linq;
using System.Collections;

namespace Process
{

   #region myClass
   public class myClass
   {
      #region Attributes
      private String xmi_type;
      private String name;
      private String visibility;
      private String myGuid;
      private Guid guid;
      private String foreignGuid;
      private int sqlId;
      private int sqlForeignKey;
      private String stereotype;
      private String stereotypeDescription;
      private String type;
      #endregion

      #region Constructores
      public myClass(String name,String visibility, String xmi_type,int sqlId)
      {
         this.name = name;
         this.visibility = visibility;
         this.xmi_type = xmi_type;
         this.guid = Guid.NewGuid();
         if (this.xmi_type == "uml:Package")
            this.myGuid = "EAPK_" + this.guid.ToString();
         else
            this.myGuid = "EAID_" + this.guid.ToString();
         this.sqlId = sqlId;
      }
```

```
public myClass(String name,int sqlId)
{
   this.name = name;
   this.visibility = "public";
   this.xmi_type = "uml:Class";
   this.guid = Guid.NewGuid();
   this.sqlId = sqlId;

   if (this.xmi_type == "uml:Package")
      this.myGuid = "EAPK_" + this.guid.ToString();
   else
      this.myGuid = "EAID_" + this.guid.ToString();
}
public myClass(String name,String visibility,int sqlId)
{
   this.name = name;
   this.visibility = visibility;
   this.xmi_type = "uml:Class";
   this.guid = Guid.NewGuid();
   this.sqlId = sqlId;

   if (this.xmi_type == "uml:Package")
      this.myGuid = "EAPK_" + this.guid.ToString();
   else
      this.myGuid = "EAID_" + this.guid.ToString();
}

#endregion

#region Get & Set

#region get
public String getName()
{
   return this.name;
}
public String getXmiType()
{
   return this.xmi_type;
}
public String getVisibility()
{
   return this.visibility;
}
public String getGuid()
{
   return this.myGuid;
}
public String getForeignGui()
{
```

```
    return this.foreignGuid;
}
public int getsqlId()
{
    return this.sqlId;
}
public string getStereotype()
{
    return this.stereotype;
}
public string getStereotypeDescription()
{
    return this.stereotypeDescription;
}
public int getSqlForeignKey()
{
    return this.sqlForeignKey;
}
public String getMyType()
{
    return this.type;
}
#endregion

#region set

public void setName(String name)
{
    this.name = name;
}
public void setXmiType(String XT)
{
    this.xmi_type = XT;
}
public void setVisibility(String visibility)
{
    this.visibility = visibility;
}
public void setStereotype(String stereotype)
{
    this.stereotype=stereotype;
}
public void setStereotypeDescription(String sDesc)
{
    this.stereotypeDescription = sDesc;
}
public void setSqlId(int s)
{
    this.sqlId = s;
}
```

```
        public void setSqlForeignID(int FK)
        {
            this.sqlForeignKey = FK;
        }
        public void setForeignGuid(String g)
        {
            this.foreignGuid = g;
        }
        public void setMyType(String type)
        {
            this.type = type;
        }
        #endregion

        #endregion

        #region Métodos
        public string getAndSetStereotypeDescription()
        {
            string returnValue="";
            switch (this.stereotype)
            {
                case "Descriptor": break;//entity.Descriptor.Description); break;
                case "identifier":  break;//entity.Identifier.Description); break;
                case "enumeration": returnValue="enumeration"; break;
                default: break;
            }
            this.stereotypeDescription = returnValue;
            return returnValue;
        }
        public void modifyGuid(int sqlRelatedEntityId,Hashtable hashElements)
        {
            if (hashElements.ContainsKey(sqlRelatedEntityId))
                this.myGuid = ((myClass)hashElements[sqlRelatedEntityId]).getGuid();
        }
        #endregion
    }
    #endregion

    public class Interoperability
    {

        public void GenerateXMI(int intProjectId)
        {
            #region Documento XMI

            XDocument docxmi = new XDocument(new XDeclaration("1.0", "windows-1252",
"yes"));

            #endregion
```

```
        #region Variables
        XNamespace xmi = "http://schema.omg.org/spec/XMI/2.1";
        XNamespace uml = "http://schema.omg.org/spec/UML/2.1";

        Hashtable hashClassEntities= new Hashtable();   //CONTIENE A LAS CLASES
CORRESPONDIENTES A LOS TAGS DE PACKAGED ELEMENT - ELEMENT
        Hashtable hashPackagedElements = new Hashtable(); //CONTIENE A LAS
DISTINTAS ETIQUETAS DE PACKAGED ELEMENT
        Hashtable hashClassElements = new Hashtable();  //CONTIENE A LAS
DISTINTAS ETIQUETAS DE ELEMENT
        Hashtable hashDiagrams = new Hashtable();
        myClass classEntities;  //CLASE CORRESPONDIENTE A LOS TAGS DE
PACKAGED ELEMENT - ELEMENT
        XElement xClassEntities;    //TAGS DE PACKAGED ELEMENT
        XElement xClassEntitiesToElement;   //TAGS DE ELEMENT
        XElement xDiagram;
        XElement xAllDiagrams;
        XElement xTags = new XElement("tags");
        #endregion

        #region System

        classEntities = new myClass("System", "public", "uml:Package", intProjectId);
        hashClassEntities.Add("System", classEntities);

        hashPackagedElements.Add("System", createPackagedElement(classEntities, xmi));
        hashClassElements.Add("System", createElement(classEntities, xmi, "Package"));

        #endregion

        #region Properties y Enumerations
        List<Entity> entities = EntitiesData.GetEntitiesWithProperties(intProjectId);

        foreach (Entity entity in entities) // se agregan todas las entidades a un hashtable
        {
           #region Desarrollo Entities

           classEntities = new myClass(entity.Description.ToString(), "public",
entity.Enumeration==true?"uml:Enumeration":"uml:Class", entity.Id);

           classEntities.setStereotype(entity.Descriptor.Descriptor==true ? "Descriptor" :
entity.Identifier.Identifier == true ? "identifier" :
entity.Enumeration==true?"enumeration":"");

           classEntities.getAndSetStereotypeDescription();

           classEntities.setSqlForeignID(((myClass)hashClassEntities["System"]).getsqlId());

           if (entity.Enumeration)
```

```
            {
               classEntities.setStereotype("Enumeration");
               classEntities.setStereotypeDescription("Enumeration");
            }

            if(!hashClassEntities.ContainsKey(classEntities.getsqlId()))
               hashClassEntities.Add(classEntities.getsqlId(), classEntities); //Agrego la clase
creada en el hash, siendo su código el ID de SQL
            if(!hashPackagedElements.ContainsKey(classEntities.getsqlId()))
               hashPackagedElements.Add(classEntities.getsqlId(),
createPackagedElement(classEntities,xmi)); //Agrego la nueva etiqueta creada al hash. Su
código es el ID de SQL

            if (hashClassElements[classEntities.getsqlId()] == null)
               hashClassElements.Add(classEntities.getsqlId(),
createElementWithTagsandAttributes(classEntities, xmi, "Class")); // Agrego la etitueta de
"Element" correspondiente a la entidad con la cual trabajo
            else
               hashClassElements[classEntities.getsqlId()] =
createElementWithTagsandAttributes(classEntities, xmi, "Class");
            #endregion
         }

         foreach (Entity entity in entities)
         {
            #region Desarrollo Enumerations
            //if (!entity.Enumeration)
            {
               foreach (EntityProperty property in entity.Properties)
               {
                  classEntities = new myClass(property.Description, "private","uml:Property",
property.Id);

                  classEntities.setSqlForeignID(property.EntityId);

                  classEntities.setStereotype(property.Descriptor == true ? "Descriptor" :
property.Identifier == true ? "identifier" : "");

                  classEntities.setStereotype(property.Stereotype);

                  classEntities.setMyType(property.getDataTypeXmi()); //Cambios: Antes:
.DataTypeDescription; Despues:.DataTypeXmi


classEntities.setForeignGuid(((myClass)hashClassEntities[classEntities.getSqlForeignKey()]
).getGuid());

                  classEntities.modifyGuid(Convert.ToInt32( property.RelatedEntityId),
hashClassEntities); //CAMBIOS: Modifica el GUID, en caso de ser necesario, para las clases
relacionadas
```

```
                xClassEntities = createOwnedAttribute(classEntities, xmi, "false");


addToPackagedElements((XElement)hashPackagedElements[classEntities.getSqlForeignKe
y()],xClassEntities);

                xClassEntitiesToElement=createAttribute(classEntities, xmi, "false");

                /*Modifico el tag de packagedValues, agregàndole el paquete correspondiente
a la clase entidad*/
                XElement
tempVar=(XElement)hashPackagedElements[classEntities.getSqlForeignKey()];
                newModifyAttributes(tempVar, xClassEntities);
                hashPackagedElements[classEntities.getSqlForeignKey()] = tempVar;

                /*Modifico el tag de element, agregàndole el attribute correspondiente a la
clase entidad*/
                XElement
tempVarII=(XElement)hashClassElements[classEntities.getSqlForeignKey()];
                newModifyAttributes(tempVarII, xClassEntitiesToElement);
                hashClassElements[classEntities.getSqlForeignKey()] = tempVarII;

                if (hashClassEntities[classEntities.getsqlId()] == null)                /*Si no
existe, lo ingreso*/
                   hashClassEntities.Add(classEntities.getsqlId(), classEntities);
                else
                   hashClassEntities[classEntities.getsqlId()] = classEntities;          /*Si
existe, lo reemplazo por la nueva instancia*/
             }
          }
          if (entity.EnumerationValues != null)
             foreach (EnumerationValue enumValue in entity.EnumerationValues)
             {
                classEntities = new myClass(enumValue.Description,
"private","uml:Property", enumValue.Id);

                classEntities.setSqlForeignID(enumValue.EntityId);

                classEntities.setStereotypeDescription("EnumerationValue");
                classEntities.setStereotype("EnumerationValue");


classEntities.setForeignGuid(((myClass)hashClassEntities[classEntities.getSqlForeignKey()]
).getGuid());

                xClassEntities = createOwnedAttribute(classEntities, xmi, "false");
```

```
addToPackagedElements((XElement)hashPackagedElements[classEntities.getSqlForeignKe
y()],xClassEntities);
                /**********************************/

                xClassEntitiesToElement = createAttribute(classEntities, xmi, "false");

                XElement tempVar =
(XElement)hashClassElements[classEntities.getSqlForeignKey()];
                newModifyAttributes(tempVar, xClassEntitiesToElement);
                hashClassElements[classEntities.getSqlForeignKey()] = tempVar;

                if (hashClassEntities[classEntities.getsqlId()] == null)
                   hashClassEntities.Add(classEntities.getsqlId(), classEntities);
                else
                   hashClassEntities[classEntities.getsqlId()] = classEntities;

             }

          #endregion
       }
       #endregion

       #region TaggedValues
       List <TaggedValueType>
taggedValues=TaggedValuesData.GetTaggedValuesTypes();
       foreach (TaggedValueType taggedValue in taggedValues)
            {

       }
       #endregion

       #region Diagrams

       String systemGuid= ((myClass)hashClassEntities["System"]).getGuid().ToString();
//GUID correspondiente a la etiqueta "System"

       foreach (DictionaryEntry element in hashClassEntities)
       {
          String tempGuid = ((myClass)element.Value).getGuid().ToString();

          if (element.Key.ToString() != "System")
          {
             xDiagram = createDiagram(tempGuid);
             if(!hashDiagrams.ContainsKey(tempGuid)) //CAMBIO NUEVO
                hashDiagrams.Add(tempGuid, xDiagram);
          }
       }

       xDiagram = new XElement("diagram",
```

```
                    new XAttribute(xmi + "id", new Guid()));

        xAllDiagrams= new XElement("diagrams",diagramsGenerator(xDiagram,
hashDiagrams, systemGuid));

        #endregion

        #region xExtension

        XElement allXElements = newElementGenerator(hashClassElements,xmi);
/*NUEVO*/
        XElement allXPackagedElements =
packagedElementGenerator((XElement)hashPackagedElements["System"],hashPackagedEle
ments);     /*NUEVO*/

        XElement xExtension = createXExtension(allXElements, xAllDiagrams, xmi);
        #endregion

        #region Profile

        XElement xProfile = new XElement(uml+"Profile");
        /*xExtension.Add(new XAttribute(xmi + "version", "2.1"),
                    new XAttribute(XNamespace.Xmlns+"uml",
"http://schema.omg.org/spec/UML/2.1/uml.xml"),
                    new XAttribute(XNamespace.Xmlns + "xmi",
"http://schema.omg.org/spec/XMI/2.1"),
                    new XAttribute(xmi + "id", "thecustomprofile"),
                    new XAttribute("name", "thecustomprofile"),
                    new XAttribute("nsPrefix", "thecustomprofile")
                    );*/
        #endregion

        #region Datos Cabecera


docxmi.Add(createHeaderData(xProfile,allXPackagedElements,xExtension,xmi,uml));

        #endregion

        //CAMBIAR PATH!!!!
        #region Guardar Documento
        try
        {
           docxmi.Save("C:\\Users\\gacevedo.GIDFIS\\Desktop\\generado.xml");
//CAMBIAR PATH
        }
        catch (Exception ex)
        {
           ex.Message.ToString();
        }
```

```
        #endregion
    }

    #region myMethods

    public XElement attributeGenerator(String name, String type, bool descriptor)
    {
        String descriptorAttribute = descriptor ? "descriptor" : "";
        XElement attribute = new XElement("attribute");
        attribute.Add(new XAttribute("name", name),
                    new XElement ("properties",
                    new XAttribute("type", type)
                            ),
                    new XElement ("stereotype",
                        new XAttribute("stereotype", descriptorAttribute)
                            )
                    );

        return attribute;
    }


    public XElement tagGenerator(String name)
    {
        XElement Tag = new XElement("tag");
        Tag.Add(new XAttribute("name", name));
        return Tag;
    }

    public XElement enumerationGenerator(String name)
    {
        XElement attribute = new XElement("attribute");
        attribute.Add(new XAttribute("name", name),
                    new XElement("stereotype",
                        new XAttribute("stereotype","enum")
                            )
                    );

        return attribute;
    }

    public XElement elementGenerator(Hashtable hashElement)    //Genero tag-etiqueta
"elements" con cada uno de sus "element"
    {
        XElement outElement = new XElement("elements");

        foreach (DictionaryEntry element in hashElement)
        {
            outElement.Add((XElement)element.Value);
        }
```

```
            return outElement;
        }

        public XElement appendTagToElement(XElement element, XElement tag)
        {
            XElement newElement = new XElement("element", new XAttribute("name",
"Componente"));
            newElement.Add(tag);

            element.LastNode.AddAfterSelf(newElement);

            return element;
        }

        public XElement modifyAttributes(XElement tree, XElement newData)
        {
            XElement pivote = (XElement) tree.FirstNode;
            pivote.Add(newData);

            tree.FirstNode.ReplaceWith(pivote);

            return tree;
        }

        public XElement modifyEnumerations (XElement tree, XElement newData)
        {
            XElement pivote = (XElement)tree.FirstNode;
            pivote.Add(newData);

            tree.LastNode.ReplaceWith(newData);

            return tree;
        }

        #endregion

        #region Métodos

        public XElement addToPackagedElements(XElement tree, XElement newData)
        {
            tree.Add(newData);
            return tree;
        }

        public XElement newModifyAttributes(XElement tree, XElement newData)
        {
            XElement pivote = (XElement)tree.LastNode;
            pivote.Add(newData);

            tree.LastNode.ReplaceWith(pivote);
```

```
        return tree;
    }

    public XElement newTagGenerator(String name, String value, Guid supraGuid)
    {
        XElement tag = new XElement("tag");

        tag.Add(new XAttribute("name", name), new XAttribute("value", value), new
XAttribute("modelElement", supraGuid), new XAttribute("xmi:id", new Guid()));

        return tag;
    }

    public XElement newElementGenerator(Hashtable hashElement,XNamespace xmi)
//Genero tag-etiqueta "elements" con cada uno de sus "element"
    {
        XElement outElement = new XElement("elements");

        XElement nuevo = new XElement("Model",new XAttribute(xmi+"idref",
"EAPK_EA4DF52E_871B_497a_B496_1F760BD42505")); /*TODO--HACERLO
APARTE*/

        outElement.Add((XElement)nuevo);

        foreach (DictionaryEntry element in hashElement)
        {
            outElement.Add((XElement)element.Value);
        }
        return outElement;
    }

    public XElement packagedElementGenerator(XElement rootPackage, Hashtable
hashElement)    //Genero tag-etiqueta "elements" con cada uno de sus "element"
    {
        XElement outElement = rootPackage;

        foreach (DictionaryEntry element in hashElement)
        {
            if(element.Key.ToString() != "System")
                outElement.Add((XElement)element.Value);
        }
        return outElement;
    }

    public XElement diagramsGenerator(XElement rootDiagram, Hashtable hash,String
systemGuid)
    {
        XElement outDiagram = rootDiagram;
```

```
        XElement elements = new XElement("elements");

        foreach (DictionaryEntry diag in hash)
        {
           if (diag.Key.ToString() != "System")
              elements.Add((XElement)diag.Value);
        }

        outDiagram.Add(new XElement("model",
                  new XAttribute("package", systemGuid),
                  new XAttribute("owner", systemGuid)),
         new XElement("properties",
                  new XAttribute("name", "System"),
                  new XAttribute("type", "Logical")),
         new XElement("style1",
                  new XAttribute("value",
"ShowPrivate=1;ShowProtected=1;ShowPublic=1;HideRelationships=0;Locked=0;Border=0
;HighlightForeign=0;PackageContents=0;SequenceNotes=0;ScalePrintImage=0;PPgs.cx=0;P
Pgs.cy=0;DocSize.cx=795;DocSize.cy=1138;ShowDetails=0;Orientation=P;Zoom=100;Sho
wTags=0;OpParams=1;ShowIcons=1;CollabNums=0;HideProps=0;ShowReqs=0;ShowCons
=0;PaperSize=9;HideParents=0;UseAlias=0;HideAtts=0;HideOps=0;HideStereo=0;HideEle
mStereo=0;ShowTests=0;ShowMaint=0;ConnectorNotation=;ExplicitNavigability=0;Advan
cedElementProps=1;AdvancedFeatureProps=1;AdvancedConnectorProps=1;ShowNotes=0;"
)),
          new XElement("style2",
                  new XAttribute("value",
"ExcludeRTF=0;DocAll=0;HideQuals=0;AttPkg=1;ShowTests=0;ShowMaint=0;SuppressF
OC=1;TDurLow=0;TDurHigh=100;TDurUnit=;TDurHide=0;TConnectorNotation=;TExplic
itNavigability=0;AdvancedElementProps=1;AdvancedFeatureProps=1;AdvancedConnectorP
rops=1;ProfileData=;MDGDgm=;ShowNotes=0;")),
          new XElement("swimlanes",
                  new XAttribute("value",
"locked=false;orientation=0;width=0;inbar=false;names=false;color=0;bold=false;fcol=0;;cls
=0;")),
                  elements);

        return outDiagram;
     }

     #endregion

     private XElement createPackagedElement(myClass classEntities, XNamespace
xNamespace) //CREA UN xELEMENT que corresponde a un packagedElement de uan
determinada Clase
     {
       XElement element = new XElement("packagedElement");
        element.Add(new XAttribute("name", classEntities.getName()),
                  new XAttribute(xNamespace + "type", classEntities.getXmiType()),
                  new XAttribute("visibility", classEntities.getVisibility()),
                  new XAttribute(xNamespace + "id", classEntities.getGuid())
```

```
                    );
        return element;
    }

    private XElement createElement(myClass classEntities, XNamespace xNamespace,
String sType) //CREA UN xELEMENT que corresponde a una etiqueta "Element"
    {
        XElement element = new XElement("element");
        element.Add(new XAttribute("name", classEntities.getName()),
                        new XAttribute(xNamespace + "id", classEntities.getGuid()),
                        new XAttribute(xNamespace + "type", classEntities.getXmiType()),
                        new XAttribute("scope", classEntities.getVisibility()),
                        new XElement("properties",
                            new XAttribute("isSpecification", "false"),
                            new XAttribute("sType", sType),
                            new XAttribute("nType", 0),
                            new XAttribute("scope", classEntities.getVisibility())
                            )
                        );
        return element;
    }

    private XElement createElementWithTagsandAttributes(myClass classEntities,
XNamespace xNamespace, String sType) //CREA UN xELEMENT que corresponde a una
etiqueta "Element", con el tag Attribute
    {
        XElement element= new XElement("element");
        element.Add(new XAttribute("name", classEntities.getName()),
                            new XAttribute(xNamespace + "idref", classEntities.getGuid()),
                            new XAttribute("scope", classEntities.getVisibility()),
                            new XAttribute(xNamespace + "type",
classEntities.getXmiType() == "uml:Enumeration" ?
"uml:Class":classEntities.getXmiType()),
                            new XElement("properties",
                                new XAttribute("sType", sType),
                                new XAttribute("scope", classEntities.getVisibility()),
                                classEntities.getStereotypeDescription()!=""?new
XAttribute("stereotype",classEntities.getStereotypeDescription()):new
XAttribute("stereotype","")
                                    ),
                            new XElement("tags"),
                            new XElement("attributes")
                            );
        return element;
    }

    private XElement createOwnedAttribute(myClass classEntities, XNamespace
xNamespace, String isDerived) //CREA UN xELEMENT que corresponde a una etiqueta
"ownedAttribute"
    {
```

```
        XElement element = new XElement("ownedAttribute");
        element.Add(
                        new XAttribute("name", classEntities.getName()),
                        new XAttribute(xNamespace + "id", classEntities.getGuid()),
                        new XAttribute(xNamespace + "type", classEntities.getXmiType()),
                        new XAttribute("visibility", classEntities.getVisibility()),
                        new XAttribute("isDerived", isDerived)
                        );
        return element;
    }

    private XElement createAttribute(myClass classEntities, XNamespace xNamespace,
String collection) //CREA UN xELEMENT que corresponde a una etiqueta "Attribute"
    {
        XElement element = new XElement("attribute");
        element.Add(
                            new XAttribute("name", classEntities.getName()),
                            new XAttribute(xNamespace + "idref", classEntities.getGuid()),
                            new XAttribute("scope", classEntities.getVisibility()),
                            new XElement("stereotype",
                                                new XAttribute("stereotype",
classEntities.getStereotype() == null ? "" : classEntities.getStereotype())),
                            new XElement("properties",
                                        new XAttribute("type", classEntities.getMyType() ==
null ? "" : classEntities.getMyType()),
                                        new XAttribute("derived", 0),
                                        new XAttribute("collection", collection))
                            );
        return element;
    }

    private XElement createDiagram(String tempGuid) //crea un xElement que corresponde
a un determinado XDIAGRAM
    {
        int left = 100, top = 0, right = 0, bottom = 0;
        XElement xDiagram = new XElement("element",
                        new XAttribute("geometry", "Left=" + left + ";Top=" + top + ";Right="
+ right + ";Bottom=" + bottom),
                        new XAttribute("subject", tempGuid),
                        new XAttribute("style", "DUID=7E4F0BBC;ImageID=0;"));

        return xDiagram;
    }

    private XElement createHeaderData(XElement xProfile, XElement
allXPackagedElements, XElement xExtension, XNamespace xmi, XNamespace uml)//crea
un xElement que contiene todos los datos necesarios que aparecen en a cabecera
    {
        return new XElement(xmi+"XMI",
                new XAttribute(xmi + "version", "2.1"),
```

```
                new XAttribute(XNamespace.Xmlns + "uml",
"http://schema.omg.org/spec/UML/2.1"),
                new XAttribute(XNamespace.Xmlns + "xmi",
"http://schema.omg.org/spec/XMI/2.1"),
                new XAttribute(XNamespace.Xmlns + "thecustomprofile",
"http://www.sparxsystems.com/profiles/thecustomprofile/1.0"),
                new XElement(xmi + "Documentation",
                   new XAttribute("exporter", "Enterprise Architect"),
                   new XAttribute("exporterVersion", "6.5")),
                 new XElement(uml+"Model",
                    new XAttribute(xmi+"type", "uml:Model"),
                    new XAttribute("name", "EA_Model"),
                    new XAttribute("visibility", "public"),
                    new XElement("packagedElement",
                       new XAttribute(xmi + "type", "uml:Package"),
                       new XAttribute(xmi + "id",
"EAPK_EA4DF52E_871B_497a_B496_1F760BD42505"),
                       new XAttribute("name", "Model"),
                       new XAttribute("visibility", "public"),
                       new XElement("packagedElement",
                          new XAttribute(xmi + "type", "uml:Package"),
                          new XAttribute(xmi + "id",
"EAPK_518D4ABD_9169_46e6_9B7D_1C612CED28E5"),
                          new XAttribute("name", "Class Model"),
                          new XAttribute("visibility", "public"),
                              allXPackagedElements,xProfile
                                    ))
                            ),
                            xExtension
                           );
      }

      private XElement createXExtension(XElement allXElements, XElement xAllDiagrams,
XNamespace xmi)//crea un xElement que contiene los datos necesarios a colocar en
"EXTENSION" (TAG Extensiòn)
      {
         XElement xExtension = new XElement(xmi + "Extension");
         xExtension.Add(new XAttribute("extender", "Enterprise Architect"),
                   new XAttribute("extenderID", "6.5"),
                   allXElements, xAllDiagrams);

         xExtension.Add(new XAttribute(xmi + "version", "2.1"),
                   new XAttribute(XNamespace.Xmlns + "uml",
"http://schema.omg.org/spec/UML/2.1/uml.xml"),
                   new XAttribute(XNamespace.Xmlns + "xmi",
"http://schema.omg.org/spec/XMI/2.1"),
                   new XAttribute(xmi + "id", "thecustomprofile"),
                   new XAttribute("name", "thecustomprofile"),
                   new XAttribute("nsPrefix", "thecustomprofile")
                   );
```

```
            return xExtension;
        }
    }
}
```