

Universidad Castilla-La Mancha

Escuela Superior de Ingeniería Informática

Departamento de Sistemas Informáticos

Programa Oficial de Postgrado en Tecnologías Informáticas Avanzadas

Trabajo Fin de Máster

INTEGRACIÓN DE TÉCNICAS DE INGENIERÍA INVERSA EN EL DESARROLLO DE INTERFACES DE USUARIO DIRIGIDO POR MODELOS

Julio 2012

Alumno Reynaldo José Cruz Ocampo

Director: Dr. D. Francisco Montero Simarro

Co-Director: Dr. D. Víctor López Jaquero

Contenido

Índice de Ilustraciones

Índice de Tablas

Introducción

Las interfaces de usuario representan una de las partes más importantes y determinantes del desarrollo de cualquier producto software. La interfaz de usuario es la cara de la aplicación ante los usuarios, que interactúan con ella; dirige al usuario en la recogida de datos para que éstos sean procesados y, al mismo tiempo, permite la visualización del procesamiento de estos datos, convirtiéndolos en información útil para la toma de decisiones.

El diseño de interfaces de usuario de calidad se preocupa por el logro de una interacción adecuada entre el usuario y el ordenador, asegurando de esta manera que el usuario pueda realizar las tareas de una manera efectiva, eficiente y satisfactoria. Las propuestas de diseño y desarrollo de interfaces de usuario de calidad se han realizado desde la disciplina de la Interacción Persona-Ordenador (IPO)

En muchas ocasiones es necesario realizar modificaciones a las interfaces de usuario por muy bien diseñadas que estas estén, recordemos que las necesidades de los sistemas cambian con el tiempo, así como también de los usuarios que interactúan con ellas. Si agregamos a lo anterior la aparición de nuevas tecnologías, que ofrecen mayores oportunidades de crecimiento y competitividad, también se hace necesaria la migración a otras plataformas de software, por lo que las interfaces de usuario tienen que poder ser redefinidas para adaptarlas a nuevos entornos y nuevas necesidades.

Ante esas necesidades hay varias soluciones posibles, la primera de ellas es coordinar con el proveedor del sistema, en caso de que éste exista, y plantearle los nuevos requisitos. En caso de que el desarrollador de la aplicación ya no brinde soporte para la misma o en su defecto que simplemente ya no exista, y tampoco se cuente con la documentación sobre el desarrollo de la aplicación nos enfrentamos ante un verdadero problema. El desarrollo de una aplicación desde cero podría llevar meses de trabajo y elevados costes de desarrollo. Además, también se debe considerar el tiempo de entrenamiento de los usuarios para que aprendan a manejar las funcionalidades del nuevo sistema y la resistencia al cambio que puedan presentar los usuarios y los problemas de interacción que puedan surgir con el resto de las aplicaciones con las que podrían compartir datos. La migración a un nuevo sistema es un problema grande por los obstáculos y los costes que presenta.

La ingeniería inversa o reverse engineering puede ayudar a resolver este tipo de problemas, como parte de un proceso de mantenimiento, mejora o re-documentación del sistema, específicamente de las interfaces de usuario en nuestro caso. De esta manera se podrían obtener las funcionalidades del sistema actual de tal manera que su comportamiento pueda ser conservado en la medida de lo posible. Pero para poder obtener un buen resultado en el proceso inverso de desarrollo de interfaces de usuario es necesario hacer uso de herramientas y metodologías sistemáticas y que aseguren el éxito. Algunas de estas propuestas se han realizado desde la Ingeniería de Software. Una de esas herramientas que puede aportar la Ingeniería de Software en este proceso inverso de desarrollo de interfaces de usuario son los modelos.

Desde hace más de una década está vigente la propuesta de usar modelos declarativos para el desarrollo de interfaces de usuario. Capturando de esta manera todas aquellas características y comportamientos que puedan ayudar a la generación automática de una interfaz de usuario.

En este trabajo de investigación, y en esta propuesta de Tesis Doctoral, se plantea el uso de técnicas de ingeniería inversa en conjunción con el desarrollo de interfaces de usuario basado en modelos para proponer una solución a los problemas mencionados anteriormente.

Capítulo 1
Asignaturas Cursadas

El presente capitulo realiza una descripción del trabajo realizado en cada una de las asignaturas que se cursaron en el periodo de formación académica del programa de Doctorado en Tecnologías Informáticas Avanzadas. Las asignaturas descritas en este capítulo se corresponden con las asignaturas cursadas durante el curso 2011/2012.

Los cursos se realizaron en la Escuela Superior de Ingeniería Informática en el Campus de Albacete.

Durante ese periodo se han cursado las siguientes asignaturas:

- Sistemas Inteligentes Aplicados a Internet
- Introducción a la Programación de Arquitecturas de Altas Prestaciones
- Tecnologías de Red de Altas Prestaciones
- Calidad de Interfaces de Usuarios: Desarrollo Avanzado
- Modelado y Evolución de Sistemas
- Tecnología de Software Orientada a Objetos

Todas las asignaturas mencionadas tienen asignados 5 créditos, por lo que representan el 50 % de la carga académica total del Máster. El otro 50% corresponde al Trabajo de Fin de Máster, que tiene asignados los créditos restantes, 30 créditos. El presente documento está asociado al Trabajo Fin de Máster.

Seguidamente se describirán las distintas asignaturas mencionadas y cursadas. A la hora de describir cada una de las asignaturas se utilizarán dos secciones. Una sección estará dedicada a describir cada asignatura, identificando profesores y contenidos, y una segunda sección describirá los trabajos realizados durante el período lectivo.

1.1 Tecnologías de Red de Altas Prestaciones

1.1.1 Descripción de la Asignatura

La Asignatura Tecnologías de Red de altas prestaciones está orientada a la enseñanza de los diferentes métodos y técnicas relacionadas con la interconexión de computadoras y la comunicación de datos a gran escala entre ellas.

La asignatura ha sido impartida por las Doctoras Dña. María Blanca Caminero y Dña. Carmen Carrión y el Doctor D. José Duato. Los contenidos presentados en esta asignatura estuvieron divididos en 4 temas y 2 seminarios.

El primer tema consistió en una introducción a las redes de altas prestaciones, por lo que permitió conocer los términos y conceptos básicos relacionados con las mismas, por ejemplo cloud, grid, cluster, etc.

El segundo tema profundizo en las arquitecturas de red que dan soporte a las principales supercomputadoras.

En el tercer tema se dedicó a la administración de los recursos en entornos distribuidos, así como a la evaluación de la calidad de los servicios de red.

El cuarto y último tema estuvo dedicado a la gestión reactiva de los recursos de red.

En los dos seminarios organizados en esta asignatura se tuvo la oportunidad de interactuar con herramientas como gridsim; una aplicación que simula el comportamiento de un entorno grid y globus, que permite la configuración de ordenadores para que interactúen en entornos grid.

1.1.2 Trabajo Realizado

El trabajo final de la asignatura consistió en la investigación relacionada con ANEKA CLOUD[1], una solución basada en .NET para la implementación de computación en la nube. El principal objetivo de dicha herramienta es tratar de lograr el mayor aprovechamiento posible de grupos de equipos conectados en entornos empresariales. La mayor parte de los ordenadores en los entornos empresariales ejecutan alguna versión de Windows como sistema operativo, en cambio la mayor parte de las aplicaciones dedicadas a la implementación de computación en la nube están desarrolladas para sistemas operativos Unix o Linux.

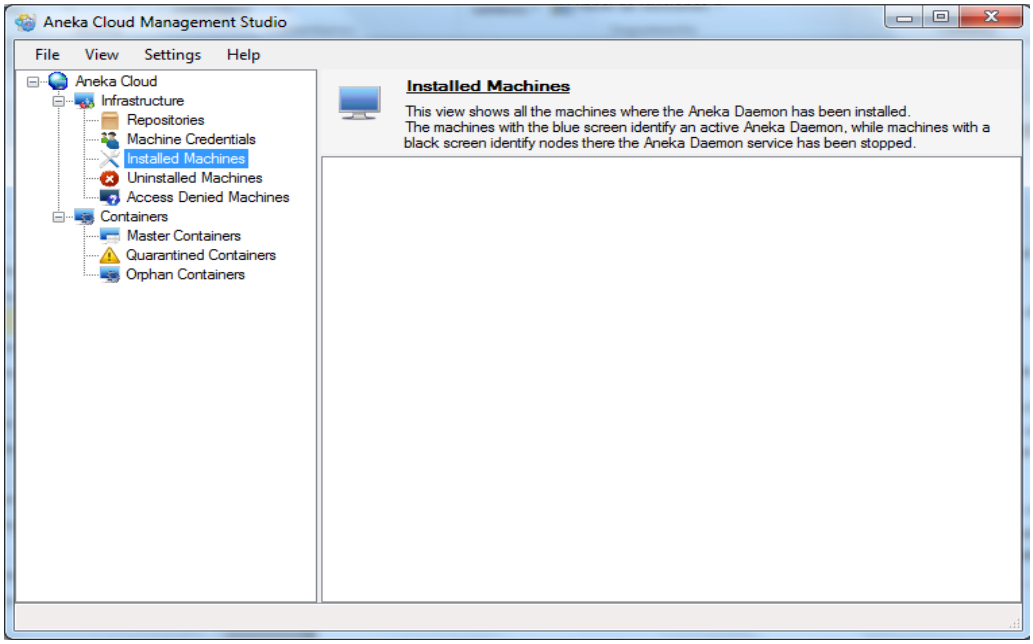

Ilustración 1-1 Aneka Cloud Management Studio

La Ilustración 1-1 muestra la interfaz correspondiente a Aneka Cloud Management Studio en cual permite poder agregar equipo de una forma fácil y rápida a la nube, también tiene soporte para agregar maquinar virtuales de Amazon Elastic Compute Cloud (Amazon EC2)[2], lo que permite agregar mayor capacidad de computo a la nube sin necesidad de adquirir equipo físico, lo que se traduce en ahorro de dinero al evitar comprar hardware.

[1] http://www.manjrasoft.com/products.html
[2] http://aws.amazon.com/es/ec2/

1.2 Introducción a la Programación de Arquitecturas de Altas Prestaciones

1.2.1 Descripción de la Asignatura

La asignatura fue impartida por los Doctores D. Diego Cazorla, D. Juan José Pardo y D. Enrique Arias. El principal objetivo del curso consistió en presentar e identificar las principales ventajas que hay detrás de la programación paralela. Para ello se utilizaron distintas herramientas software que soportan dicha actividad de programación.

Para convencer sobre las bondades y diferencias existentes entre la programación secuencial y la paralela se utilizaron ejemplos relacionados con la manipulación y operación con matrices de gran tamaño.

Entre las herramientas presentadas en la asignatura están:

- MPI, que es una interfaz de paso de mensajes que es usada en aplicaciones para que exploten los múltiples procesadores que pueda tener un ordenador.
- Blacs, Blas y Scalapack, que proveen librerías útiles para realizar operaciones enfocadas a resolver problemas en algebra lineal.

Cursando esta asignatura también ejecutamos aplicaciones distribuidas en clúster.

1.2.2 Trabajo Realizado

El trabajo final del curso estuvo enfocado al estudio y análisis de patrones de diseño útiles para resolver problemas relacionados con la programación paralela. Específicamente se estudió la colección de patrones OPL (Our Pattern Lenguaje)[3]. OPL fue diseñado con el objetivo de resolver y documentar los problemas que se presentan de manera frecuente en programación paralela. También sirve como una guía para que los futuros programadores paralelos puedan disponer de conocimiento acerca de las soluciones más habituales y sobre las herramientas que se usan en la solución de problemas en programación paralela.

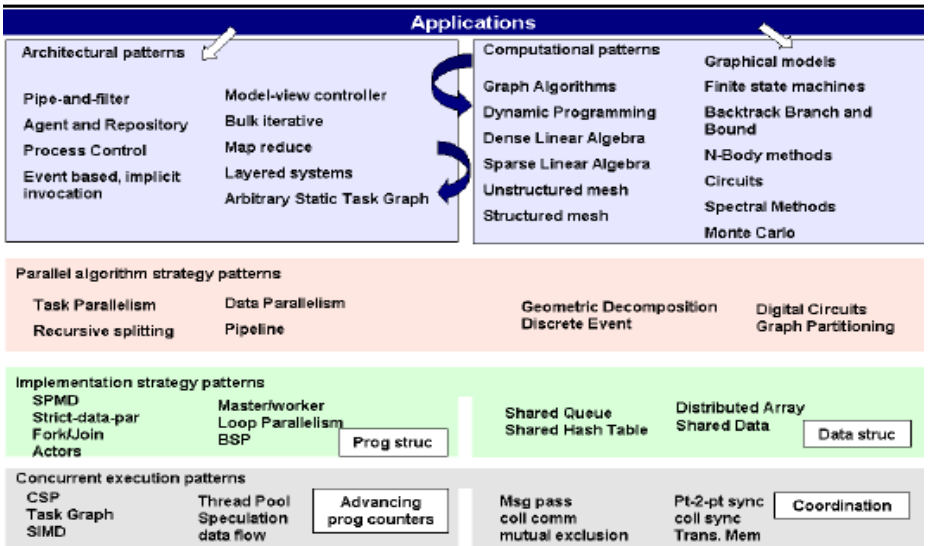

Ilustración 1-2 Estructura de OPL

[3] http://parlab.eecs.berkeley.edu/wiki/_media/patterns/opl_pattern_language-feb-13.pdf

La Ilustración 1-2 muestra la estructura en la que conforman los patrones OPL, las capa de aplicación se subdivide en dos secciones que son los patrones estructurales y los patrones computacionales; ambos relacionados íntimamente, y proveen estrategias de soluciones a problemas relacionados con la arquitectura de las aplicaciones paralelas y la forma en que se debe ejecutar los procesos. La segunda capa describe patrones que resuelven problemas relacionados con la implementación de algoritmos paralelos, la capa siguiente describe la forma como esos algoritmos son implementados y la ultima capa proporciona estrategias de ejecución de aplicaciones paralelas.

El estudio de estos patrones de diseño puede proveer a los diseñadores de aplicaciones paralelas de los conocimientos necesarios para diseñar aplicaciones que aprovechen de forma adecuada las prestaciones de los ordenadores.

1.3 Sistemas Inteligentes Aplicados a Internet

1.3.1 Descripción de la Asignatura

La asignatura fue impartida por los Doctores Dña. María Julia Gallego, D. Ismael García y D. José Miguel Puerta.

La asignatura se dividió en 3 unidades temáticas:

La primera de esas unidades consistió en una introducción y una presentación de conceptos relacionados con las redes bayesianas, los modelos de redes probabilísticas, la independencia de nodos, el criterio de d-separación, el modelado de reyes bayesianas y las inferencias en las mismas. También en esta primera parte se organizó un seminario relacionado con los conceptos anteriores impartido por el Doctor Janos Sarbo.

En esta asignatura se hizo uso de Elvira, una herramienta que ayuda en el modelado de las redes bayesianas así como en el cálculos de probabilidades.

La segunda unidad temática profundizo en los conceptos de redes bayesianas, hicimos uso de algoritmos y métricas que pueden determinar qué modelo de red bayesiana es mejor y por qué, haciendo uso de las métricas BIC, algoritmo PC, estimaciones por máxima verosimilitud, aprendizaje paramétrico, etc.

La tercera y última unidad temática estuvo relacionada con las metaheurísticas y la minería de datos. En esta unidad se presentaron algunas técnicas usadas en la búsqueda y optimización de información, como por ejemplo los algoritmos genéticos. En esta sección también se trató el tema de los clasificadores bayesianos como el Naive Bayes, seminaive Bayes, y dDB.

1.3.2 Trabajo Realizado

Se realizaron una serie de ejercicios propuestos por los profesores responsables de cada unidad temática, cada uno de ellos estuvo enfocado a la solución de problemas relacionados con probabilidades condicionales usando para ellos las redes bayesianas. También se planteo la solución de ejercicios relacionados con aprendizaje paramétrico y métricas. En la última unidad se planteo la solución de ejercicios relacionados con algoritmos genéticos, clasificadores bayesianos y minería de datos. Para ellos con herramientas como WEKA se probó la efectividad de los clasificadores bayesianos.

1.4 Calidad de Interfaces de Usuario: Desarrollo Avanzado

1.4.1 Descripción de la Asignatura

La asignatura fue presentada por los Doctores D. Pascual J. González López, D. Francisco Montero Simarro y D. Víctor Manuel López Jaquero.

La asignatura fue dividida en 3 bloques, el primero de ellos fue impartido por D. Pascual J. González, en el cual se introdujeron algunos términos de calidad de interfaces de usuario y la percepción que se tiene de la misma. También se hiso mención de las principales características que debe tener un producto de calidad y de las diferencias que existen entre la calidad de un producto manufacturado y un producto software debido a la intangibilidad de este último, en consecuencia la percepción de calidad puede ser subjetiva. Se realizo una introducción a las interfaces de usuario.

El segundo bloque impartido por D. Víctor López Jaquero fue una introducción a los modelos, y al desarrollo de interfaces de usuario basado en modelos (MB-UIDE) y de los principales modelos involucrados en el desarrollo de interfaces de usuario, como son dominio, tareas, usuario, dialogo, interfaz abstracta, interfaz concreta y final. Además de se realizo una revisión de lenguajes los lenguajes de especificación de interfaces de usuario, y de algunas metodologías que se han usado como por ejemplo USIXML, el cual considera 4 niveles de abstracción para el desarrollo de interfaces de usuario: Tareas & Conceptos, UI Abstracta, UI Concreta y UI Final. Añadido a esto en este bloque se realizo mención de las posibles transformaciones que se pueden realizar entre los modelos: Reificación, Abstracción (Reverse), Reflexión, Traducción.

El tercer bloque estuvo a cargo de D. Francisco Montero, el cual estuvo relacionado con la usabilidad, concretamente con la especificación y evaluación, se mostro la calidad en uso desde dos puntos de vista, el primero desde el punto de vista de ingeniero de software, la cual incluye la calidad interna del producto y el proceso de calidad que se sigue para su desarrollo, el segundo punto de vista corresponde al usuario, el cual tiene una percepción de la calidad relacionado con el uso de la interfaz y con la facilidad que con que se realizan las operaciones. Se realizo también una introducción a algunos estándares relacionados con la calidad de software como son el ISO/IEC 9126-1:2001 y el ISO 9241-11. Se realizo una descripción de algunas métricas asociadas a la usabilidad. Además se mostro la forma como adaptar la usabilidad a la experiencia del usuario para lograr software de mayor calidad.

1.4.2 Trabajo Realizado

El trabajo realizado en la asignatura estuvo relacionado con los Lenguajes de especificación de transformaciones y entornos de desarrollo de interfaces de usuario, en el cual se mostro el funcionamiento de ATL(Atlas Transformation Languaje) como lenguaje de transformación de modelos usado en el MB-UIDE(Model Based- User Interface Development) y cómo es posible realizar transformaciones de abstracción usando una metodología como UsiXML, este trabajo sirvió como base para el trabajo fin de master ya que se evaluaron las ventajas de ATL como lenguaje de trasformación mediante un caso de estudio en el que se plantean transformaciones de abstracción entre un modelo de interfaz de usuario concreta a una interfaz de usuario abstracto.

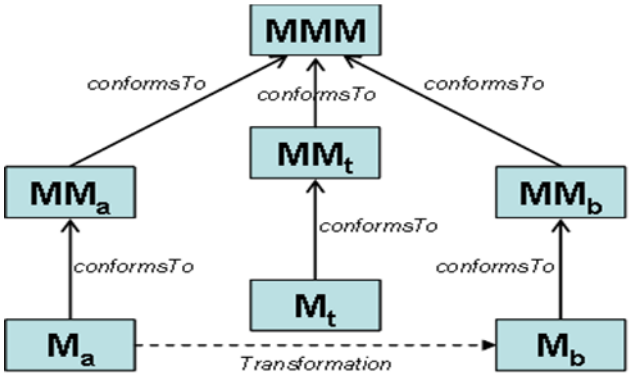

Ilustración 1-3 Esquema de Transformación ATL

ATL responde al esquema descrito en la Ilustración 1-3 donde se puede observar que hace uso del Metametameta modelo, empleado por Eclipse Modeling Framework, lo que permite el uso de Metamodelos Ecore como UsiXML.

1.5 Tecnología de Software Orientada a Objetos

1.5.1 Descripción de la asignatura

La asignatura fue impartida por los doctores Dña. Elena María Navarro, Dña. María Dolores Lozano, D. Víctor Manuel Ruiz Penichet y D. Ricardo Tesoriero, la asignatura se dividió en 3 bloques.

El primer bloque fue impartido por Dña. Elena María Navarro, se realizo una introducción al desarrollo dirigido por modelos (DSDM), de la misma forma se mostro el uso de la herramienta Eclipse Modeling Framework (EMF)[4], para la definición de metamodelos con Meta-Object Facility (MOF) el cual permite provee la facilidad de definir y extender nuevos modelos, vimos que entre las ventajas que presenta el usar EMF es la capacidad de generación de código para Java. Se realizaron transformaciones de modelo a modelo usando la especificación de la OMG QVT[5], y de modelo a texto usando para ellos XPAND.

En el segundo bloque Dña. María Dolores Lozano se realizo una descripción de lo que es el Desarrollo de Interfaces de usuario Basado en modelos (MB-UIDE). Se introdujo IDEAS como propuesta metodológica para usarse en el MB-UIDE, de describieron los modelos usados por IDEAS en el desarrollo de interfaces de usuario, se realizo un prototipo de una interfaz de usuario siguiendo esta propuesta.

El tercer bloque estuvo a cargo de D. Víctor Manuel Ruiz Penichet, en este bloque se hablo de TOUCHE(Task-Oriented and User-Centred Process Model for Developing Interfaces for Human-Computer-Human Environments) la cual se basa en el modelado de procesos para el desarrollo de interfaces de usuario en entornos colaborativos, usando para ello técnicas centradas en el usuario y en las tareas. Este bloque fue compartido con D. Ricardo Tesoriero el cual introdujo CAUSE(Desarrollo dirigido por modelos de aplicaciones sensibles al contexto para ambientes de computación ubicua). Cada una de estas metodologías soportadas cada una por una herramienta CASE que ayudan a los desarrolladores en el desarrollo de aplicaciones.

[4] http://www.eclipse.org/modeling/emf/
[5] http://www.omg.org/spec/QVT/1.0/

1.5.2 Trabajo Realizado

El trabajo realizado en esta asignatura se relaciono directamente con la transformación de modelos haciendo uso de XSLT(Extensible Stylesheet Language Transformations), el cual permite la transformación de modelos gracias a su capacidad de poder transformar un documento XML en otro documento XML, este trabajo sirvió para la comparación que entre los diferentes lenguajes de transformación que se tratan en el capítulo tercero de este trabajo.

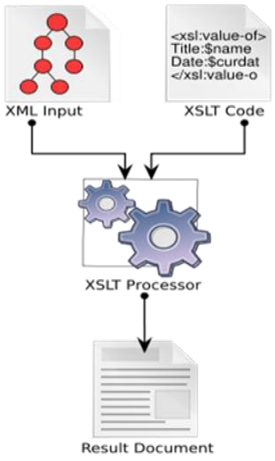

Ilustración 1-4 Esquema de Transformación XSLT

La Ilustración 1-4 describe el esquema de transformación de XSLT, donde se recibe 2 ficheros de entrada uno con el documento XML transformar y una hoja de estilos con las reglas de transformación, ambos ficheros son cargados por un procesador que genera un fichero como salida. Este esquema de transformación no hace uso de metamodelos, por lo que puede representar un problema debido a que puede generar un archivo que no sea acorde a la metodología usada en la especificación de las interfaces de usuario.

1.6 Diseño y Evaluación de Sistemas

1.6.1 Descripción de la Asignatura

La asignatura fue impartida por los Doctores D. Rafael González Casado, D. Luis Orozco Barbosa y D. Aurelio Bermúdez Marín, la asignatura fue desarrollada en 3 bloques:

En el primer bloque impartido por D. Rafael González Casado, se realizo una introducción a la evolución y modelado de sistemas, se definieron los conceptos generales relacionados con la asignatura. Se mostro la importancia de la simulación del comportamiento de un sistema, así como los elementos que intervienen en el proceso de simulación; como por ejemplo como debe realizarse la selección de los datos de entrada, como se recolectan los resultados. El proceso de simulación nos permite poder obtener una visión anticipada del comportamiento del sistema antes de desarrollarlo, por lo que es posible la identificación de errores, cuellos de botellas que se puedan presentar en el mismo.

El segundo bloque estuvo a cargo de D. Luis Orozco, en el cual se trataron temas como la evaluación de las prestaciones de un sistema mediante el uso de teoría de colas, en este bloque se realizaron ejercicios prácticos orientados a evaluar las prestaciones de un sistema en

donde están presentes los conceptos de líneas de espera (cola), servidores, tiempo de servicio para poder calcular los tiempos de espera y de servicio de cada elemento en la línea de espera.

El tercer bloque impartido por D. Aurelio Bermúdez con el que se realizaron simulaciones con los simuladores NS-2 y OPNET Modeler, se realizaron simulaciones sobre redes Ethernet, redes simples, LAN conmutadas, enlaces Wireless, se realizaron evaluaciones con los datos recogidos por las simulaciones y se nos enseño la forma correcta de analizar los resultados obtenidos.

1.6.2 Trabajo Realizado.

El trabajo final de la asignatura fue un trabajo de investigación acerca del uso de las cadenas de Markov en el modelado y evolución de los sistemas. Las cadenas de Markov son muy útiles en la evaluación y simulación de sistemas ya que permiten evaluar los posibles estados de un sistema, donde la posibilidad que un estado cambie depende únicamente de las probabilidades del estado actual y no de los anteriores.

Tomando como ejemplo la Ilustración 1-5 vemos que tiene 4 estados (0,1,2,3) las flechas indican las posibles transiciones entre los mismos, la probabilidad que sede una transición del estado 1 al estado 2 depende únicamente de la información proporcionada por el estado 1, la información de los demás estados es irrelevante.

Ilustración 1-5 Ejemplo de estados de un Sistema

Se desarrollaron varios ejemplos para poder mostrar la utilidad de las cadenas de Markov en el modelado y la evaluación de sistemas, un ejemplo de uso diario de las cadenas de Markov es el PageRank de Google.

Capítulo 2

Estado del Arte

2.1 Introducción

Un modelo de un sistema se puede definir como la descripción simplificada o la especificación parcial de un sistema y su entorno para un propósito determinado. Un modelo se representa utilizando un conjunto de elementos gráficos y textos (OMG, 2003). Limbourg (2004) define un modelo como una vista simplificada de los objetos del mundo real.

Los modelos representan una de las mejores guías para el desarrollo de software, ya que nos permiten identificar las distintas vistas que ofrece un sistema o producto software. Los modelos permiten una mejor interpretación de las especificaciones y ayudan continuamente al ingeniero de software en la documentación de los sistemas a lo largo del proceso de desarrollo. También contribuyen, indirectamente, al correcto mantenimiento de los mismos.

Como constatamos, el concepto de modelo será un elemento clave en nuestra propuesta de investigación y en él descansarán nuestras propuestas. Previamente, en este capítulo identificaremos qué modelos están presentes de manera asidua en el desarrollo de interfaces de usuario. Así, este capítulo está dividido en tres secciones; la primera de ellas estará relacionada con las interfaces de usuario, en las que se identifican definiciones, usos, tipos de interfaces de usuario. La segunda sección del capítulo se hace referencia al desarrollo de interfaces de usuario utilizando técnicas basadas en modelos (Mb-UIDE). En este sentido, se hará mención a las propuestas metodológicas que se han realizado en ese campo, se revisarán los lenguajes de especificación propuestos y las herramientas necesarias para el desarrollo de interfaces de usuario. La tercera, y última, parte estará relacionada con la Ingeniería inversa y la interacción. En ella se describen conceptos de ingeniería inversa, así como algunas herramientas usadas en este ámbito, también se hará mención de distintas aproximaciones que se han propuesto por parte de la ingeniería inversa en el ámbito de las interfaces de usuario.

2.2 Interfaces de Usuario e interacción

Las interfaces de usuario proporcionan un método de interacción entre los usuarios y los ordenadores. Su objetivo es facilitar la forma en que los usuarios proporcionan instrucciones al ordenador, es decir actúa como un mediador entre el ordenador y el usuario; traduciendo las ordenes que el usuario da al ordenador a un lenguaje que la maquina pueda entender y del lenguaje de la máquina a datos que el usuario pueda interpretar fácilmente.

Durante las últimas dos décadas se han realizado muchos avances en el desarrollo de las interfaces de usuario. Éstas han pasado de ser muy complejas como en el caso de las interfaces de líneas de comando que tenían el “obstáculo” de la memorización de comandos y la curva de aprendizaje que esta memorización imponía a usuarios novatos, a las interfaces gráficas de usuario (GUI), que facilitan las interacción por el uso del ratón y menús desplegables. Más recientemente, la interacción puede presentarse de muchas otras formas, por ejemplo a través de interfaces de usuario táctiles, graficas, textuales, etc. Seguidamente, en el siguiente

apartado, se realizara una breve descripción de algunos de los estilos de interacción más extendidos y utilizados. Esta descripción identificará desde las interfaces textuales a las interfaces de voz, pasando por las interfaces gráficas y táctiles.

2.2.1 Interfaces textuales

Las interfaces de texto fueron la primera forma de interacción existente entre el usuario y el ordenador. En estas interfaces el usuario interactúa utilizando comandos para proporcionar instrucciones al ordenador. El principal problema de estas interfaces está en que el usuario tiene que memorizar una serie de comandos con sus parámetros correspondientes para indicarle al ordenador qué hacer en cada momento.

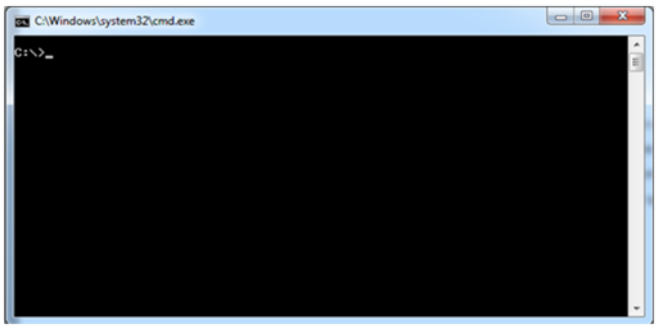

Ilustración 2-1 Pantalla Símbolo del Sistema de Windows

La Ilustración 2-1 representa el simbolo del sistema del en un sistema operativo Windows. La mayoría de los sistemas operativos siguen manteniendo interfaces de texto como en el caso del símbolo del sistema Windows o las terminales de Linux.

2.2.2 Interfaces Graficas

Las interfaces graficas de usuario o GUI (*Graphical User Interface)* son lo opuesto a las interfaces de texto. En este tipo de interfaces se presentan un conjunto de objetos y símbolos gráficos que permiten mejorar la interacción entre el usuario y el ordenador. Los ejemplos más conocidos de interfaces de usuario graficas son las ventanas, estas se encuentran presentes en la mayor parte de los sistemas operativos actuales. También son conocidas como WIMP (*Windows Icons Menus Pointer*). Los principales aliados de las interfaces de usuario graficas son los dispositivos de entrada, como por ejemplo el ratón.

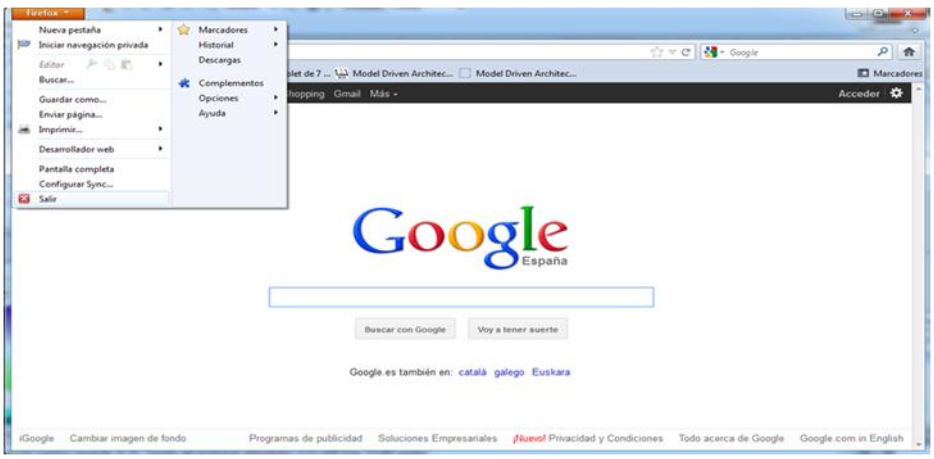

Ilustración 2-2 Interfaz Grafica de Usuario

La Ilustración 2-2 es un buen ejemplo de interfaces WIMP. En ella, el contenedor principal de los elementos GUI es la ventana, las órdenes o instrucciones del usuario se realizan a partir del menú principal que se encuentra desplegado, esta interfaz hace uso del ratón para seleccionar y especificar órdenes del menú principal.

2.2.3 Interfaces táctiles

Las interfaces táctiles consisten en el uso del sentido del tacto o mediante el uso de un apuntador permite dar órdenes o introducciones al dispositivo u ordenador.

El uso de las interfaces de usuario táctiles se ha extendido mucho en los últimos años, un ejemplo muy común está en los cajeros automáticos, los puntos de venta, muchas maquinarias industriales y los teléfonos más recientes. La interacción táctil están soportadas por los sistemas operativos más usados en la actualidad: Windows, Linux, MacOS; aunque también se han desarrollado sistemas operativos especialmente diseñados para funcionar con este tipo de interfaz: Android, Windows Mobile, iOS, Windows Phone, etc.

La Ilustración 2-3 muestra el menú principal de Android, su interfaz es llamada TouchWiz, la cual fue desarrollada por Samsung Electronics y sus socios. Esta interfaz es el complemento al sistema operativo Android como lo es Aero a Windows o Gnome y KDE a Linux.

Ilustración 2-3 Menú Principal Android

2.2.4 Interfaces de voz

En las interfaces de usuario por voz el ordenador recibe instrucciones por parte del usuario por medio de la voz, para poder inicializar procesos o servicios. Estas interfaces aún presentan complicaciones ya que se requiere hablar claramente para que el ordenador entienda lo que el usuario dice. Por otro lado, la utilización de estas interfaces requiere algo de paciencia por parte del usuario que, en ocasiones, debe repetir varias veces las instrucciones hasta que el sistema las interpreta de manera adecuada.

En definitiva, y después de identificar diferentes estilos de interacción entre el usuario y el ordenador también hay disponibles distintas formas de abordar el desarrollo de interfaces de usuario. Uno de los más extendidos y aceptados dentro de la Comunidad de desarrolladores

está el desarrollo basado en modelos, que será el que presentaremos y describiremos a continuación.

2.3 Desarrollo basado y dirigido por modelos

Una de las propuestas que más auge está teniendo en las últimas décadas es el desarrollo de aplicaciones dirigidos por modelos, también conocida con el nombre de MDD (Model Driven Development). En las propuestas bajo este epígrafe se contempla el uso de los modelos para algo más que una simple documentación. Con el MDD se logra reducir la complejidad al desarrollar aplicaciones para múltiples plataformas; esto es posible debido a que el MDD logra aumentar el nivel de abstracción por medio de la especificación de modelos, obteniendo de esta manera una independencia de la plataforma de destino en los niveles de abstracción más altos. MDD persigue, igualmente, los objetivos de interoperabilidad, portabilidad y reusabilidad a la hora de abordar el desarrollo del software.

En el ámbito concreto del desarrollo de interfaces de usuario diferentes técnicas, esta vez bajo la denominación de entornos de desarrollo basados en modelos (Mb-UIDE) viene utilizándose incluso antes del concepto MDD.

2.3.1 El desarrollo de interfaces de usuario basado en modelos

Con el propósito de automatizar la generación de interfaces de usuario a mediado de los años noventa surgió la propuesta de usar modelos para el desarrollo de interfaces de usuario.

El desarrollo de interfaces de usuario basado en modelos consiste en la utilización de modelos en los cuales se consideran las diferentes etapas del desarrollo de la interfaz de usuario.

Las técnicas Mb-UIDE proponen un conjunto de abstracciones, metodologías y herramientas que permiten hacer del desarrollo de interfaces de usuario un proceso consistente e ingenieril. Los modelos son representados gráficamente, normalmente usando herramientas basadas en el lenguaje unificado de modelado (UML), que permiten la representación de los distintos elementos que conformarían la interfaz de usuario en distintos niveles de abstracción. Cuanto más alto sea el nivel de abstracción conseguido mayor será la independencia de la plataforma en el diseño de una interfaz de usuario, permitiendo, de esta manera, realizar todas las especificaciones del diseño dejando de lado los detalles técnicos relacionados con la plataforma o la modalidad de la interacción.

Con MB-UIDE se plantea el desarrollo de la interfaz de usuario como una tarea paralela al desarrollo de la aplicación en su conjunto, por lo que claramente se hace una separación de la especificación de la interfaz de usuario del desarrollo del sistema en su totalidad del que forma parte.

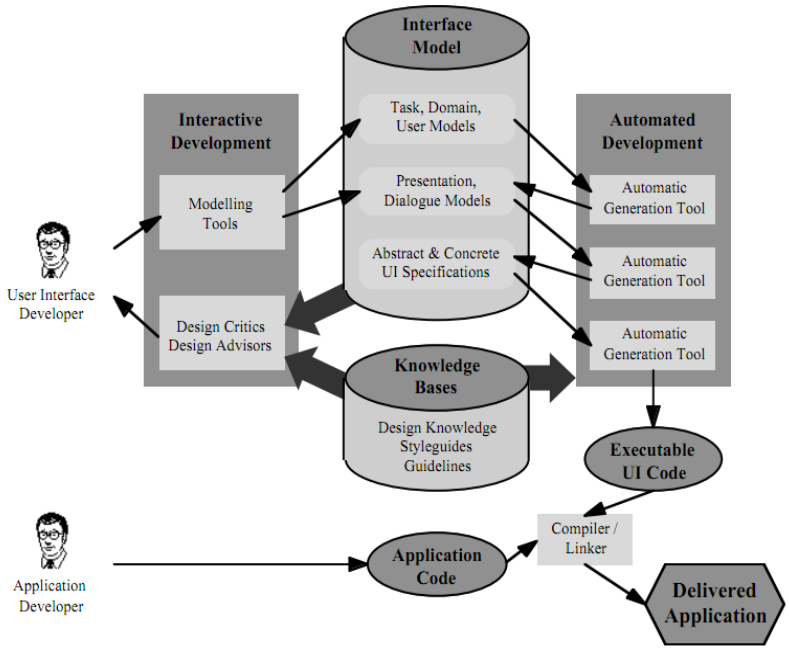

Ilustración 2-4 Esquema general asociado a un entorno de desarrollo de interfaces de usuario basado en modelos

(fuente: Schlungbaum, 1996)

El esquema mostrado en la Ilustración 2-4, describe los actores, actividades y artefactos considerados en las primeras propuestas bajo el concepto de Mb-UIDE. Además, también se muestra la relación con el desarrollo general de un producto software, que estaría considerado en la parte inferior del esquema.

En los apartados siguientes se realiza una descripción de los principales modelos que fueron considerados inicialmente en el desarrollo de interfaces de usuario. Estos modelos son los de tareas, el de dominio, el de presentación, el de dialogo y el de usuario.

2.3.1.1 Modelo de dominio

El dominio describe los objetos que se manipulan a través de la interfaz. Parte de sus funciones son identificar y catalogar componentes de software que se pueden aplicar dentro del entorno de la aplicación.

Un modelo de dominio puede ser representado mediante un diagrama entidad-relación y diagramas de clases (véase Ilustración 2-5). En ellos se identifican las principales entidades manipuladas (las cuales identifican un objeto del mundo real inequívocamente) existentes en un sistema, así como la interrelación entre ellas y las propiedades que puedan tener.

La Figura 2.5 muestra la representación que conforma el dominio mediante el uso de un diagrama de clases, con sus atributos y operaciones asociados.

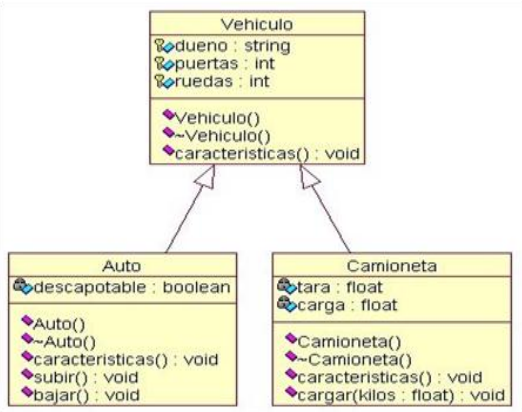

Ilustración 2-5 Diagrama de Clases

2.3.1.2 *Modelo de tareas*

En el modelo de tareas se identifican las tareas que podrán ser realizadas por el usuario utilizando el sistema. El modelo de tareas ayuda a entender cómo el usuario interactúa con el sistema y también contribuye a la identificación de los datos que serán manipulados.

Para la descripción de una actividad se suele recurrir a la utilización de una descomposición jerárquica de la misma mostrando las tareas secuenciales o concurrentes que deben realizarse hasta completarla. Los modelos de tareas son útiles para identificar el tiempo y la precedencia con la que se deben ejecutar las tareas.

En la actualidad y en Mb-UIDE, existen muchas notaciones que son utilizadas en el modelado de tareas. Una de las notaciones más conocidas y extendidas es ConcurTaskTrees (Paternò, 1997) en la que describe una estructura jerárquica de las tareas; que viene a ser una representación en forma de árbol (véase la Figura 2.6).

La Ilustración 2-6 muestra un ejemplo de modelo de tareas usando la notación CTT (ConcurTaskTrees) .

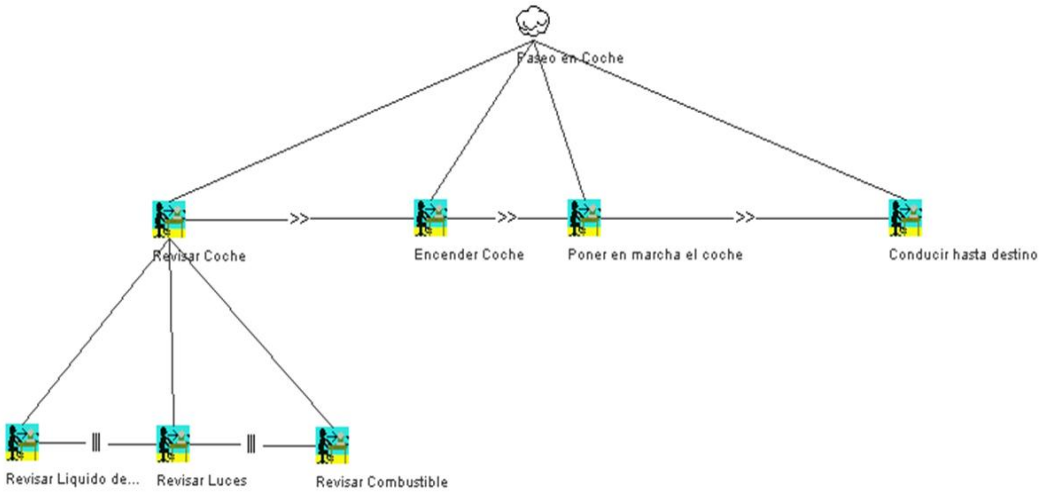

Ilustración 2-6 Descripción jerárquica de un modelo de tareas utilizando la notación CTT

2.3.1.3 Modelo de Presentación

En el modelo de presentación se realiza la descripción de la interfaz con la que interactuara el usuario, es decir la interfaz de usuario final, por lo que detalla cada uno de los componentes que conforman la interfaz de usuario. En otros casos y propuestas se proponen distintos niveles en la especificación de interfaces de usuario (véase (Limbourg Q. , Multi-Path Development of User Interfaces, 2004))

Uno de los resultados del proyecto Cameleon[6] fue la propuesta y el establecimiento de distintos niveles en la especificación del modelo de presentación. En dicha propuesta, entre otros modelos y en lo que a modelo de presentación se refiere, se habló de modelo de presentación a nivel abstracto, concreto y final.

La propuesta realizada desde el proyecto Cameleon permite la separación entre objetos abstractos y objetos concretos de interacción, y permite que la interfaz pueda especificarse independientemente de la plataforma donde se ejecutará.

2.3.1.4 Modelo de Dialogo

El modelo de dialogo describe la comunicación que se realiza entre el hombre y la maquina, detalla cuándo el usuario puede ingresar información, cuándo se deben seleccionar datos, y cuándo el sistema mostrará información procedente de la realización del computo.

Puede ser representado mediante diagramas de estado, diagramas de secuencia o diagramas de transición. Con la evolución de las técnicas relacionadas con los entornos Mb-UID este tipo de diagramas ha quedado incluido dentro del modelo de tareas y ha perdido, de este modo, su identidad como modelo independiente.

2.3.1.5 Modelo de usuario

El modelo de usuario aporta en los entornos de Desarrollo basados en Modelos un apartado para considerar las características concretas y específicas de los usuarios que interactúan. Dichos usuarios pueden presentar características concretas que deben tenerse en cuenta. El modelo de usuario aporta esta facilidad.

2.3.2 Propuestas metodológicas en Mb-UID

Conforme se ha ido desarrollando el concepto MB-UIDE, han aparecido distintas propuestas metodológicas de cómo llevar a cabo el proceso de desarrollo de interfaces de usuario, entre ellas deseamos destacar las siguientes propuestas: **IDEAS** (**I**nterface **D**evelopment **E**nvironment within o**AS**is) (Lozano, Gonzalez, & Ramos, 2001), MOBI-D (Puerta, 1997), TERESA (Santoro, 2005), MasterMind (Szekely, Sukaviriya, Castells, Muthukumarasamy, & Salcher, 1995), Wisdom (Jardim Nunes & Falcão e Cunha, 2000), UMLi (Pinheiro da Silva & W. Paton, 2003) y, más recientemente, USIXML (Limbourg, 2004).

Otra propuesta Mb-UIDE interesante la constituye **MasterMind** (Szekely, Sukaviriya, Castells, Muthukumarasamy, & Salcher, 1995) a diferencia de IDEAS que proporciona una especificación del modelo de tareas y el modelo de dominio. En MasterMind el diseñador tiene que crear el modelo de tareas, el modelo de dominio de la aplicación y el modelo de presentación. El modelo de la aplicación se especificaba usando CORBA. El modelo de tareas

[6] Proyecto Cameleon. http://giove.isti.cnr.it/projects/cameleon.html

describe las tareas del usuario final usando para ello una estructura jerárquica de tareas. El modelo de presentación describe la estructura de la interfaz de usuario incluyendo los elementos estáticos (controles) y dinámicos (información procesada en tiempo de ejecución).

MOBI-D (Puerta, 1997) es un entorno altamente interactivo que representa todos los aspectos relevantes del diseño de una interfaz de usuario usando para ellos modelos declarativos como el modelo de tareas de usuario, dialogo y presentación. La describe la arquitectura de MOBI-D, en ella se muestra la separación distintiva que aporta Mb-UIDE, y que consiste en separar el proceso de diseño de la interfaz de usuario del resto del código de la aplicación.

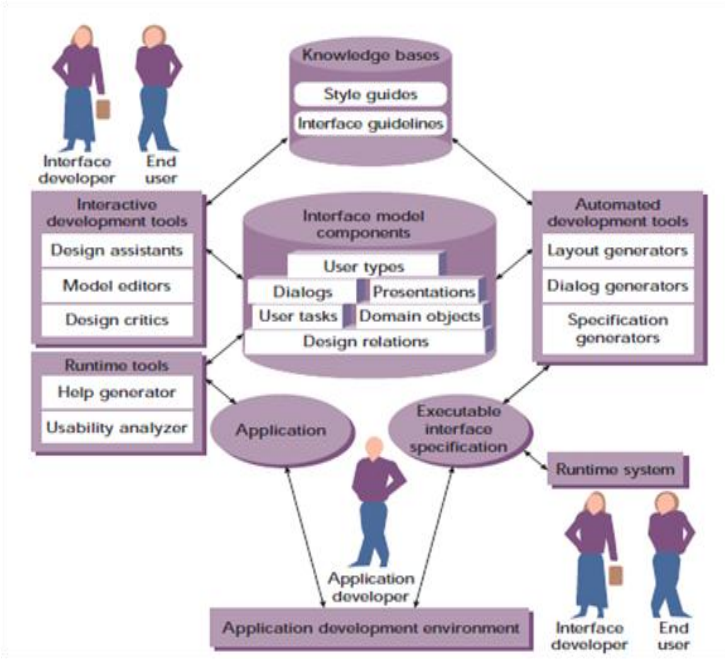

Ilustración 2-7 Arquitectura MOBI-D

El ciclo de vida de desarrollo de MOBI-D está conformado por:

- Elicitación de Tareas
- Modelado de Tareas y Modelado de Dominio
- Integración de Dominio y Tareas
- Diseño de la presentación y dialogo
- Pruebas del Usuario

Dentro de la literatura también encontramos a ***Wisdom*** (Jardim Nunes & Falcão e Cunha, 2000) es una propuesta para el desarrollo de interfaces de usuario basadas en modelos, es una extensión de UML. En Wisdom se propone el uso de estereotipos relacionados con cada una de las etapas de la especificación de la interfaz. Así, para el modelo de presentación se proponen los siguientes estereotipos:

- **Navegación:** se usa para representar la interacción entre dos clases, denotando el movimiento del usuario de un espacio de navegación a otro.
- **Contains:** es una asociación entre dos clases de espacio de interacción, denotando la clase origen(contenido) y la clase destino(content).

- **Input Element:** representa la información recibida por el usuario y la información que puede manipular.
- **Output Element:** representa la información que es presentada al usuario, la información que recibe este elemento no es manipulable.
- **Action:** representa algo que el usuario puede hacer en la interfaz de usuario que pueda producir cambios en el estado interno del sistema.

IDEAS es un enfoque orientado a objetos para el desarrollo de interfaces soportado por OASIS (Open and Active Specification of Information Systems) y la metodología –OOMethod, con el que cubre el ciclo de vida del sistema de análisis, diseño e implementación. El objetivo principal en IDEAS es la generación automática de interfaces de usuario con un enfoque orientado a objetos, objetivo que puede ser cumplido gracias a la especificación de distintos modelos declarativos. Esta propuesta tiene una clara inspiración en técnicas ingenieriles y basadas en UML. IDEAS usa los siguientes modelos declarativos:

- Modelos de Tareas
- Modelo de Dominio
- Modelo de Usuario
- Modelo de Dialogo
- Modelo de Presentación

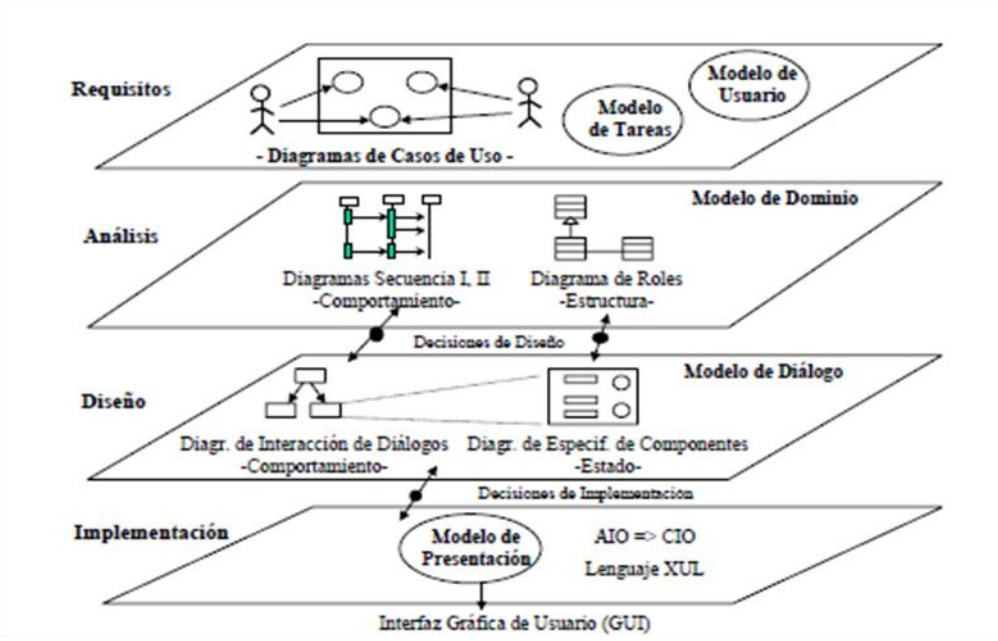

Ilustración 2-8 Ciclo de Vida de Sistemas IDEAS

La Ilustración 2-8 muestra gráficamente la propuesta IDEAS; sus etapas, los modelos y los artefactos que se usan en cada etapa. El modelo de tareas y el modelo de usuario en IDEAS son definidos en la etapa de requisitos del cicdo de vida de desarrollo. El modelo de dominio es definido en la etapa de análisis y el modelo de dialogo en la etapa de diseño. Además en esta etapa está presente el modelo de presentación.

UMLi (Pinheiro, 2002) es una ampliación de las notaciones propuestas en UML para resolver la limitaciones que presenta UML para el diseño y especificación de interfaces de usuario. Al

estar basada en UML conserva todas las características y posibilidades de ese lenguaje UMLi propuso seis componentes de interacción abstracta para la representación de la interfaz de usuario:

- ***FreeContainers***: se renderiza como un cubo con bordes punteados, se encuentra en el nivel más alto de interacción, no puede ser contenido por ninguna otra clase.
- ***Containers:*** se dibuja como un cilindro, en él se agrupan diferentes objetos de interacción.
- ***Inputters:*** se representan como triangulo con punta hacia abajo, reciben información proveniente de los usuarios.
- ***Editors:*** se representa mediante un rombo, puede ser Inputter y Displayer de acuerdo a al contexto en el que se encuentre.
- ***Displayers:*** se dibuja como un triangulo con la punta hacia arriba, su función es la de mostrar información al usuario.
- ***ActionInvokers:*** se dibuja como una flecha apuntando hacia la derecha, recibe instrucciones directas del usuario.

USIXML (Limbourg Q. , Multi-Path Development of User Interfaces, 2004) esta basada en el marco del proyecto, europeo Cameleon, usa 4 niveles de abstracción para la definición de los modelos:

- Tareas & Conceptos
- Interfaz de Usuario Abstracta
- Interfaz de Usuario Concreta
- Interfaz de Usuario Final

Una de las principales ventajas que supone el uso de UsiXML es que a nivel abstracta especifica la interfaz en términos de contenedores abstractos de interacción y componentes individuales de interacción, cada componente individual de interacción tiene asociada una faceta relacionada con el tipo de interacción que pueda tener un componente con el usuario y con el ordenador.

Todas las propuestas mencionadas anteriormente, que se comparan en la Tabla 2.1, hacen énfasis en lo que es el desarrollo de interfaces de usuario con un enfoque hacia adelante. Pero poca o escasa referencia se hace en ellas a su uso en la aplicación de técnicas de ingeniería inversa. Únicamente, la propuesta usiXML dispone de algunas herramientas que contemplan alguna trayectoria inversa en el desarrollo de interfaces de usuario, como veremos en secciones posteriores.

También debemos destacar que las propuestas mencionadas anteriormente hacen uso de lenguajes de especificación para poder dar vida a la interfaz de usuario final que se genera a los largo de las transformación entre modelos. En el apartado siguiente se hará mención a algunos de los lenguajes de especificación.

Propuesta	Modelos	Notación	Ciclo Completo de Desarrollo	Considera la ingeniería inversa	Referencia
MasterMind	Tareas Dominio Presentación		NO		(Szekely, Sukaviriya, Castells, Muthukumarasamy, & Salcher, 1995)
MOBI-D	Tareas Dominio Usuario Dialogo Presentación	MIMIC	SI	NO	(Puerta, 1997)
TERESA	Tareas Presentación	CTT AUI	NO	NO	(Berti, Correani, Paterno, & Santoro, 2004)
Wisdom	Dominio Tareas Presentación	UML	NO	NO	(Jardim Nunes & Falcão e Cunha, 2000)
IDEAS	Tareas Dominio Usuario Dialogo Presentación	UML XUL	SI	NO	(Lozano, Gonzalez, & Ramos, 2001) (Lozano, Gonzalez, & Ramos, 2001)
UMLi	Presentación	UML		NO	(Pinheiro, 2002)
USIXML	Tareas Dominio AUI CUI Contexto Transformación	USIXML	SI	Si, parcialmente	(Limbourg Q. , Multi-Path Development of User Interfaces, 2004)

Tabla 2.1. Comparativa de distintas propuestas metodológicas asociadas al desarrollo basado en modelos de interfaces de usuario.

2.3.3 Lenguajes de especificación ligados a Mb-UID

Ligados a las propuestas de desarrollo basados en modelos, comentadas en la sección anterior, han aparecido y se han desarrollado distintos lenguajes y notaciones para dar soporte a la especificación y necesidades concretas que requiere el desarrollo de interfaces de usuario. En el ámbito del desarrollo de interfaces de usuario se pueden encontrar muchos lenguajes de especificación. En este apartado identificaremos y analizaremos algunos de ellos y describiremos algunas de las características que poseen. Concretamente las principales notaciones que comentaremos serán las basadas en XML y dentro de ellas, UIML, TeresaXML, CTT, XUL y usiXML centrarán nuestra atención.

XML(*Extensible Markup Languaje*) (W3C, W3C)es uno de los lenguajes de especificación más conocidos y muchas de las propuestas de lenguajes de especificación de interfaces de usuario están basadas en XML. XML es un estándar avalado por el consorcio World Wide Web (W3C).

XML es un lenguaje de meta-marcas, lo que significa que es un lenguaje que no tiene un conjuntó finito de etiquetas lo que garantiza su extensibilidad y permite que los diseñadores y desarrolladores puedan agregar elementos de acuerdo a las necesidades que se presenten durante las etapas de desarrollo de aplicaciones. También suele describir los programas que procesan. Los archivos XML contienen texto y no código binario por lo que se puede abrir con cualquier editor de texto

Se debe aclarar que XML no es un lenguaje de programación, tampoco cuenta con compiladores que produzcan código ejecutable relacionado con XML, en cambio si se puede utilizar para guardar la configuración que puede ser leída por aplicaciones compiladas o interpretadas.

Las especificaciones XML al poder ser leídas desde cualquier editor de texto cuenta con la ventaja de tener independencia del sistema operativo en que se use. Las características con las que cuenta XML han hecho posible que a partir de él han surgido lenguajes de especificación basados en XML que permiten aprovechar todas sus características en el desarrollo de interfaces de usuario.

UIML (User Interface Markup Language) (Abrams, Phanouriou, Batongbacal, Williams, & Shuster) es un lenguaje descriptivo basado en XML para el desarrollo de interfaces de usuario, surge como una idea para crear una notación universal para la especificación de interfaces de usuario para múltiples dispositivos, la idea era desarrollar una herramienta extensible que pudiera separar la especificación de la interfaz del código que no estuviera relacionado con ella. Describe la interfaz de usuario en cinco secciones que son las descripciones, estructuras, datos, estilos y eventos, la Ilustración 2-9 describe el esqueleto de una especificación de interfaz en UIML.

```
<?xml version="1.0" standalone="no"?>
    <uiml version="2.0">
        <interface name="Figure5" class="MyApps">
            <description>...</description>
            <structure>...</structure>
            <data>...</data>
            <style>...</style>
            <events>...</events>
        </interface>
        <logic> </logic>
    </uiml>
```

Ilustración 2-9 Estructura de UI en UIML

XUL (Mozilla) es un lenguaje de interfaces de usuario desarrollado por Mozilla. Este lenguaje está basado en XML por lo que XUL cuenta con todas las ventajas y características de XML que hemos mencionado con anterioridad, por lo que la portabilidad de las interfaces de usuario descritas usando XUL está garantizada. Además, XUL permite la creación de la mayoría de los controles usados en las interfaces de usuario modernas, algunos de los elementos que se pueden crear son cajas de texto, barras de herramientas con botones, menús, atajos de teclados. Puede hacer uso de HTML, XML, CSS y JavaScript, para mejorar la presentación y para el manejo de eventos que se produzcan. XUL, por las características mencionadas con anterioridad, ha sido utilizado en distintas propuestas para poder crear modelos de presentación a nivel concreto, por ejemplo en IDEAS.

La Ilustración 2-11 muestra una ventana generada mediante el uso de XUL y la Ilustración 2-10 muestra la especificación de la ventana mencionada anteriormente.

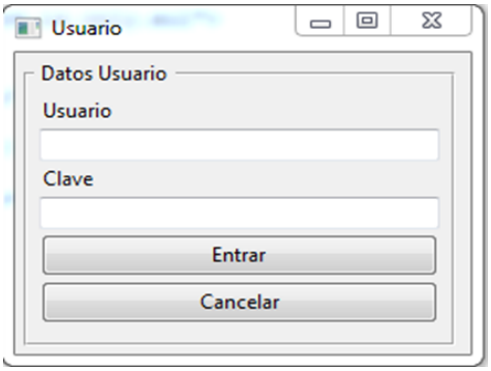

Ilustración 2-11 Ventana Generada con XUL

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
<window id="Usuario"
      title="Usuario"
      orient="horizontal"
      xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
    <groupbox width="250" height="200">
        <caption label="Datos Usuario"/>
        <label control="{buddy control id}"  value="Usuario"/>
        <textbox value=""/>
        <label control="{buddy control id}"  value="Clave"/>
        <textbox value=""/>
        <button orient="horizontal" id="find-button" label="Find" width="50"/>
        <button id="cancel-button" label="Cancel"/>
    </groupbox>
</window>
```

Ilustración 2-10 Especificación de la Ventana

Aunque XUL no fue desarrollado específicamente para dar soporte al MB-UIDE, sus características permiten que se pueda usar como lenguaje de especificación, en el caso de IDEAS (Lozano, Gonzalez, & Ramos, 2001) (Lozano, Gonzalez, & Ramos, 2001) es usado para renderizar la interfaz de usuario Final.

Siguiendo con la línea de los lenguajes de especificación identificamos **TERESAXML** (Berti, Correani, Paterno, & Santoro, 2004), que es un lenguaje de especificación que da soporte a TERESA (***T**ransformation **E**nvironment for inte**R**activ**E** **S**ystems represent**A**tions),* una

herramienta para generar interfaces de usuario partiendo de la especificación del modelo de tareas usando la notación ConcurTaskTrees (CTT) (Paternò, 1997).

Con TERESAXML es posible especificar modelos de tareas y especificaciones de la presentación a nivel abstracto. Con TERESA es posible generar estas especificaciones y lograr generar de manera automática especificaciones de interfaz a nivel concreto y final en distintas plataformas. TERESA permite dar soporte a las siguientes transformaciones:

- De modelo de tareas a interfaz de usuario abstracta
- De UI abstracta a UI concreta
- De UI concreta a distintas interfaces de usuario final en distintas plataformas y modalidades (desktop, móvil, voz, etc.)

Para desarrollar la propuesta presentada en este trabajo de investigación se hará uso de **USIXML** ***(Us**er **I**nterface e**X**tensible **M**arkup **L**anguage)*** (Limbourg Q. , Multi-Path Development of User Interfaces, 2004) es un lenguaje de especificación de interfaces de usuario basadas en modelos, además de una propuesta metodológica, como comentamos en el apartado anterior. UsiXML, hace posible la generación de interfaces de usuario en múltiples plataformas. (USIXML). Es un lenguaje de marcado basado en XML que describe la interfaz de usuario en múltiples contextos de uso como las Interfaces de Caracteres, las Interfaces Graficas de Usuario, Interfaces de Usuario de Auditoria. En otras palabras, aplicaciones interactivas con diferentes técnicas de interacción, modalidades de uso y plataformas de comunicación pueden ser descritas en USIXML. Esta propuesta es un refinamiento de los resultados obtenidos del proyecto de investigación Cameleon (Technologies), el cual estructura el cido de vida de desarrollo de interfaces de usuario en 4 niveles de abstracción. El nivel de abstracción más alto los constituyen las Tareas y los conceptos (Domino) que se corresponde con el Modelo Independiente de la computación CIM en el Desarrollo Dirigido por Modelos (OMG, 2003). En el siguiente nivel nos encontramos con la Interfaz de usuario Abstracta, correspondiente al Modelo Independiente de la Plataforma (PIM) en MDE. Le sigue la Interfaz de usuario concreta correspondiente al modelo especifico de la plataforma (PSM). Y en el último nivel de abstracción propuesto en CAMALEON encontramos la interfaz de usuario final, mismo que se corresponde con el nivel de código en el desarrollo di rigido por modelos.

USIXML permite trabajar con los diferentes niveles abstracción propuestos en CAMELEON, a continuación se realiza una breve descripción de los cuatro niveles de abstracción propuestos:

- **Tareas & Dominio**: describe las tareas que deberán ser ejecutadas por el usuario interactuando con el sistema y los objetos que son requeridos para que las tareas sean ejecutadas, los objetos de dominio son asociados a las clases.
- **Interfaz de Usuario Abstracta**: provee una especificación de la interfaz de usuario en términos de objetos abstractos de interacción (AIO). Los objetos abstractos de interacción pueden ser de dos tipos: componentes abstractos individuales (AIC) y contenedores abstractos (AC). Los AIC por la forma como se presentan permiten realizar una descripción independiente de la plataforma de los componentes
- **Interfaz de Usuario Concreta**: realiza una especificación en términos de objetos concretos de interacción, es una abstracción de la Interfaz de usuario final tratar de proporcionar un lenguaje independiente de la plataforma en la medida de lo posible.

- **Interfaz de usuario Final**: es la interfaz de usuario operacional, ejecutándose sobre una plataforma computacional especifica.

Ilustración 2-12 Niveles Abstracción USIXML

USIXML es la única propuesta que contempla la ingeniería inversa con herramientas como REVERSIXML, que será descrita en el apartado siguiente al estar dedicado a las herramientas usadas para dar soporte a las propuestas Mb-UIDE.

2.3.4 Herramientas que soportan los Mb-UIDE

Relacionado con las distintas propuestas, lenguajes y notaciones comentados hasta el momento existen distintas herramientas para dar soporte al uso de las mismas. En este apartado mencionaremos aquellas más destacadas.

UIDE (User Interface Design Enviroment) (Foley, Kim, Kovacevic, & Murray, 1986) es una herramienta que permite el diseño de interfaces de usuario con un nivel de abstracción elevado, recoge una descripción conceptual de la interfaz; es decir se debe realizar una descripción del modelo de la aplicación (modelo de dominio) que consiste en las acciones de la aplicación, acciones de la interfaz y las técnicas de interacción. El modelo de la aplicación también consiste en un conjunto de tareas que serán ejecutadas pro usuarios finales, también deben ser incluidas sus limitaciones operativas y los objetos en sobre los que deben actuar esas tareas.

AME Y JANUS son muy similares desde el punto de vista de los modelos declarativos, ambos sistemas enfatizan en la generación automática de la interfaz de usuario desde un modelo de dominio extendido orientado a objetos. Durante la generación automática de la interfaz ambos sistemas hacen uso de múltiples bases de conocimiento. Ambas generan interfaces de usuario para múltiples lenguajes de programación como por ejemplo C++.

ConcurTaskTrees Editor (HIIS Laboratory) es la herramienta que da soporte a la metodología de notación de tareas del mismo nombre propuesta por (Paternò, 1997), es un entorno para la edición de análisis de modelo de tareas, es muy útil en el desarrollo de aplicaciones que comienzan con el análisis de las tareas que realiza el usuario. **TERESA** (Berti, Correani, Paterno, & Santoro, 2004) es una herramienta para el desarrollo de interfaces de usuario para múltiples dispositivos, considera tres niveles de abstracción: modelo de tareas, interfaz de usuario abstracta e interfaz de usuario concreta. El nivel más bajo de abstracción (la interfaz de usuario concreta) es dependiente de la plataforma. Es basado en XML por lo que se puede usar

cualquier lenguaje que derive de XML para la creación de los modelos. Provee interfaces XHTML para escritorios, dispositivos móviles y VoiceXML.

Existe un conjunto de herramientas que estas asociadas a USIXML. Por ejemplo, **REVERSIXML** (Bouillon, Reverse Engineering of Declarative User Interfaces, 2006) es una herramienta de ingeniería inversa que puede obtener a partir de una página web HTML su especificación en USIXML, tanto la interfaz abstracta como la interfaz concreta con el fin de redirigir la plataforma de destino de la interfaz de usuario existente a otra. El objetivo de ReversiXML es realizar el proceso de ingeniería inversa de un sitio web y obtener modelos concretos y abstractos.

Algunas otras herramientas ligadas también a la propuesta UsiXML son, **GrafiXML** (Michotte & Vanderdonckt, 2008) que permite la edición especificaciones concretas de interfaces de usuario y el modelado del contexto, es capaz de generar automáticamente el código de interfaces para plataformas XHTML,HTML, XUL, etc. **TransfomiXML** (Limbourg & Vanderdonckt, 2004) es otra herramienta que permite la transformación de modelo a modelo usando grafos, permite la creación, modificación, almacenamiento de especificaciones de interfaces de usuario. Otras herramientas asociadas a usiXML se recogen gráficamente en la Ilustración 2-13

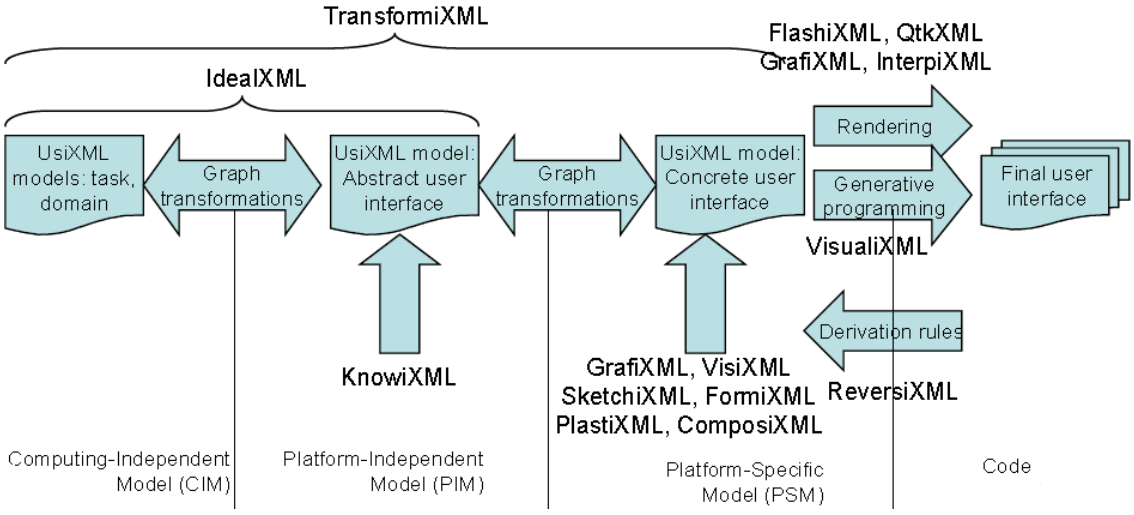

Ilustración 2-13 Herramientas asociadas y alcance de las mismas ligadas a la propuesta usiXML

2.4 Ingeniería Inversa e Interacción

Las interfaces de usuario comprenden una de las partes más importantes de cualquier aplicación. La mayoría de las aportaciones relacionadas con el desarrollo de interfaces de usuario contempla enfoques ingenieriles hacia adelante, donde se parte de niveles de abstracción altos (dominio y tareas), y se avanza hasta la interfaz de usuario final.

Los entornos Mb-UIDEs son claros ejemplos de aplicación de ingeniería hacia adelante, por ejemplo IDEAS (Lozano, Gonzalez, & Ramos, 2001) contempla los modelos a los largo de todo el ciclo de vida del desarrollo de sistemas. Pero qué sucede en los siguientes escenarios:

- ¿qué sucede cuando tenemos un sistema en funcionamiento y se deben hacer cambios a interfaz del mismo, ya sea para adaptarlas a las nuevas necesidades de los usuarios o cambiar de plataforma?
- ¿Qué hacer cuando no se cuenta con documentación ni soporte de la aplicación en funcionamiento?
- ¿Cómo debemos proceder cuando no se el lenguaje de programación en el que fue desarrollado ha caído en desuso y no se encuentra disponibilidad de desarrolladores que lo usen?

En el ámbito de este trabajo la respuesta a esas interrogantes estarán dirigidas hacia la consideración y aplicación de técnicas de ingeniería inversa.

Tomando en cuenta que el 48 % del código de una aplicación corresponde a la implementación la interfaz de usuario, también hay que señalar que dicho código es dependiente de la plataforma, por lo que si se pudiese obtener la especificación de la interfaz de usuario en un modelo abstracto se podría facilitar su mantenimiento y se podrían realizar futuras implementaciones; es en este punto donde interviene el Mb-UIDE.

En este mismo contexto, la propuesta metodológica USIXML, que cuenta con un conjunto de modelos para la especificación de la interfaz de usuario a lo largo de diferentes niveles de abstracción como se menciono en apartados anteriores es una de las propuestas y notaciones que tomaremos como punto de partida para especificar y soportar el desarrollo inverso de interfaces de usuario. En el marco de trabajo de usiXML se ha identificado trazabilidad tanto hacia adelante como hacia atrás entre las diferentes transformaciones de modelos, por lo que usiXML es una herramienta a priori idónea para la aplicación de ingeniería inversa.

El uso de la ingeniería inversa en el ámbito del desarrollo de interfaces de usuario combinada con los modelos contemplados en Mb-UIDE tendría como resultado una serie de ventajas, partiendo desde la más importante que sería la obtención de la especificación de la interfaz de la aplicación, la cual podría conducir a la generación de una nueva interfaz sin importar la plataforma de destino de la misma. Implícita a la ventaja anterior nos encontramos con la portabilidad, que se deriva por del nivel de abstracción en el que se presentan los modelos de tarea, dominio e interfaz abstracta. Otra ventaja que se tendría seria la posibilidad de documentación de la aplicación, debido a los modelos que pudiesen ser generados en un proceso de ingeniería inversa basado en modelos.

2.4.1 Alcance de la Ingeniería Inversa

La Ingeniería Inversa se puede definir de distintas formas, una definición reciente de dicho concepto se corresponde con el proceso por el que un artefacto, dispositivo o software construido haciendo uso de métodos ingenieriles, es deconstruido con el objetivo de revelar sus detalles internos, como son su diseño y su arquitectura. (Eilam, 2005).

El uso de la Ingeniería Inversa no es nuevo, se ha aplicado desde hace mucho tiempo, por ejemplo con la revolución industrial cuando un producto nuevo e innovador era presentado en el mercado por una empresa, los competidores de la misma se enfrentan ante la curiosidad de saber cómo fue construido dicho producto, para ello aplican técnicas de ingeniería inversa

con el objetivo de descubrir los secretos que se esconden detrás del producto, para poder recrear sus características y generar uno nuevo con características similares o superiores.

La ingeniería Inversa se usa principalmente para tres fines:

- Duplicar y producir partes de un fabricante de piezas originales cuando el diseño no está disponible.
- Para reparar o remplazar partes dañadas de un equipo cuando no se tiene conocimiento o no se dispone del diseño de las mismas.
- Para generar un modelo o prototipo basados en una parte existente para el análisis.

Tradicionalmente, el termino ingeniería inversa ha estado ligado únicamente a lo que son productos manufacturados, pero en las últimas décadas se ha comenzado a utilizar en la industria informática para el desarrollo de software. El uso de la Ingeniería inversa en el desarrollo de software se define como el proceso de análisis de un sistema para crear representaciones del sistema a un nivel de abstracción más alto (Chikosfsky & Cross, 1990).

Cuando se habla de niveles a abstracción más altos se hace referencia a los conceptos y a los requisitos que conducen al desarrollo y a la implementación del software. Bajo este punto de vista la ingeniería inversa considera una serie de transformaciones desde los niveles de abstracción más bajos (implementación) hasta los niveles de abstracción más altos (requisitos), recuperando de esta manera la arquitectura de la aplicación.

La Ilustración 2-14 muestra la relación existente entre la ingeniería directa (forward) y la ingeniería inversa (reverse).

Ilustración 2-14 Relación Ingeniería directa e Ingeniería Inversa

2.4.2 Herramientas que soportan la Ingeniería Inversa

Para llevar a cabo el proceso de ingeniería inversa existen varias herramientas que sirven de apoyo en el proceso, entre ellas podemos encontrar: descompiladores, desensambladores y herramientas de carácter general basadas en UML. Seguidamente revisaremos todos estos grupos de herramientas.

2.4.2.1 Descompiladores

Los descompiladores son programas que obtienen el código fuente de un programa compilado, puede ser usado para la detección de errores, debido a su principal característica que es la recuperación del código fuente.

En el mercado se pueden encontrar descompiladores para una gran gama de lenguajes de programación: para Java podemos encontrar JDec (JDec), que tiene la capacidad de descompilar archivos .dass y .jar. También existen múltiples descompiladores de archivos Flash con los que es posible acceder a los recursos multimedia utilizados para su elaboración.

No se han encontrado en la revisión llevada a cabo, sin embargo, descompiladores que permitan obtener las descripciones de las interfaces de usuario asociadas a los productos software.

2.4.2.2 Desensambladores

Los desensambladores son herramientas que toman como entrada un archivo ejecutable y proveen como salida un archivo que contiene el código en lenguaje de ensamblado correspondiente a la aplicación. La diferencia con el conjunto de herramientas que aglutinan los descompiladores está en el nivel de abstracción al que trabajan; aquellos a nivel de lenguaje de alto nivel, mientras que éstos a nivel de lenguaje ensamblador.

En esta categoría podemos encontrar a Interactive Disassembler(IDA) (Hex-Rays, 2012), MSIL(Microsoft Intermediate Language) es una herramienta complementaria del ensamblador MSIL; extrae información a partir de un archivo Ejecutable Portable (PE).

2.4.2.3 Herramientas Basadas en UML

Muchas de las herramientas CASE que se usan hoy en día tienen soporte para ingeniería inversa hasta cierto grado. La gran mayoría de esas herramientas se limitan a proporcionar un diagrama de clases, sin embargo no proporcionan una especificación de cómo se relacionan esos diagramas con el código de la aplicación. Entre esas herramientas podemos encontrar los siguientes ejemplos reseñables:

- **Together** (Borland Software Corporation, 2007) que soporta ingeniería inversa para aplicaciones desarrolladas en C++, C#, Java, VB, etc. En el caso de que el código fuente esté escrito en Java tiene la capacidad de poder generar un diagrama de clases UML al mismo tiempo que se obtiene el código fuente.
- **Rational Rose** (IBM) da soporte a ingeniería inversa en aplicaciones desarrolladas en Java y en C++. Cuando el código origen es Java construye una estructura de árbol que contiene las clases, interfaces y asociaciones encontradas en los niveles más altos. Los métodos y miembros son anidados debajo de las clases propietarias.
- La herramienta de ingeniería inversa **IDEA** (Abstract Implementation and Design with the Unified) (Kollmann & Gogolla, Application of UML Associations and Their Adornments in Design Recovery, 2001) fue desarrollada con el propósito de la re documentación de las aplicaciones desarrolladas en Java, para ello hacen uso de UML, para ello se desarrollo un meta modelo para el lenguaje Java, los modelos del programa bajo estudio se almacenan como instancias de estructuras de datos que se

corresponden con el metamodelo. Se usa un Framework de traducción para obtener el modelo UML a partir del modelo de Java.

2.5 Trabajos relacionados con ingeniería inversa e interacción

Relacionado más específicamente con la línea de investigación que propondremos en este documento podemos encontrar trabajos como la Ingeniería inversa de páginas Web descrito en (Bouillon, Limbourg, Vanderdonckt, & Michotte, 2005), donde se utilizan conceptos de ingeniería inversa para obtener la especificación concreta de una interfaz de usuario a partir del código HTML de una página web. La herramienta que da soporte a este proceso se denomina ReversiXML (Bouillon, Limbourg, Vanderdonckt, & Michotte, 2005).

En ReversiXML se hace uso de reglas de derivación, representadas en forma de funciones. El objetivo de las reglas de derivación es poder identificar patrones de lenguaje del código fuente y de esa manera poder crear la estructura correspondiente en términos de un grafo de interfaz de usuario. Las transformaciones entre modelos son llevadas a cabo mediante transformación por grafos, aunque existen otras alternativas que serán comentadas más adelante.

Otro ejemplo significativo es **Celleste Project** (Stroulia, El-Ramly, Kong, & Sorenson, 1999). Celleste es una herramienta que trata de entender la información y la lógica que presenta un sistema mediante el uso de ingeniería inversa. Se especializa en sistemas basados en transacciones desarrollados para MainFrames cuyo propósito es soportar la entrada de datos y las tareas. Este proyecto tiene una serie de herramientas para realizar el proceso de ingeniería inversa. Estas herramientas son capaces de obtener las tareas y el dominio de la aplicación por medio de un seguimiento de las acciones realizadas por el usuario. Una vez obtenido el modelo de tareas puede generar una especificación abstracta de la interfaz de usuario. Una de las desventajas que se presenta en esta propuesta es que es orientada a sistemas de gestión AS/400 por lo que no sería directamente aplicable a otras plataformas.

Las dos propuestas anteriores están limitadas a un lenguaje concreto lo que las convierte en técnicas dependientes de la plataforma.

En la revisión bibliográfica realizada, hemos identificado una relación entre el uso de técnicas de ingeniería inversa e interfaces de usuario y el objetivo de soportar actividades de prueba. Así, en (Banerjee & Nagarajan, 2003) se propone "GUI Ripping", el cual se define como un proceso dinámico en el cual las GUI que componen la aplicación es recorrida extrayendo los widgets (objetos GUI) que componen la aplicación así como sus respectivas propiedades y valores. La información extraída es verificada por el "test –designer" y usada para generar casos de prueba automáticamente. Los algoritmos propuestos para la extracción de la información de los widgets actúan sobre interfaces de usuario gráficas Java y Microsoft Windows. El objetivo de la aplicación de ingeniería inversa en el GUI Ripping no es obtener documentación, ni la aplicación de reingeniería, el objetivo es soportar la realización de pruebas del producto software con el objetivo de encontrar errores.

Relacionado con la anterior propuesto, en (Coimbra Morgado & Pascoal Faria, 2011) se propone la obtención de la interfaz de usuario usando para ello métodos de especificación, esta investigación forma parte de un proyecto llamado AMBER iTest, el cual está orientado a la realización de pruebas. Para la extracción de la información que permita realizar el proceso de

ingeniería inversa desarrollaron una aplicación llamada ReGUI, la cual obtiene información cuando la aplicación a la cual se le quiere aplicar ingeniería inversa se encuentra en ejecución. ReGUI es capaz de obtener información acerca del comportamiento de la aplicación, por ejemplo los elementos de navegación (menús) y los estados del mismo (cuando están habilitados o deshabilitados) y los cambios de estado que se producen debido s a la interacción.

Algunos trabajos que se están realizando en este mismo ámbito son el trabajo de (MUHAIRAT, AL-QUTAISH, & ATHAMENA, 2011), que en propone capturar todos los componentes (controles) de la interfaz grafica de usuario y almacenarlos en un tabla temporal no normalizada, una vez que se han capturado los elementos se procede a la normalización lo que genera una nueva tabla, cuando esta tabla ha sido generada se procede a convertirla a un diagrama de clases. En (Sánchez Ramón, Sánchez Cuadrado, & García Molina, 2010) se propone la reingeniería de interfaces de usuario de sistemas legados haciendo uso de MDA. Una de las actividades más importantes que proponen es la identificación de las relaciones existentes entre los elementos (controles) que componen la interfaz grafica de usuario, la propuesta se centra en el análisis de aquellas interfaces de usuario desarrolladas en entornos de desarrollo rápido de aplicaciones (RAD).

2.6 Análisis y conclusiones

Las Interfaces de Usuario constituyen la principal forma de comunicación que existe entre el usuario y el ordenador. Existe una marcada diferencia entre las interfaces de usuario que se usaban tan sólo hace unos años y las actuales. Fruto de la familiarización con la evolución de las interfaces de usuario hemos identificado diferentes estilos de interacción y tipos, los que han evolucionado desde textuales, como en el caso de MS-DOS a la interfaces Graficas compuestas por ventanas, botones, iconos y menús (GUI) hasta otras interfaces más intuitivas y donde se busca la comunicación directa e invisible entre el usuario y la máquina.

También se ha constatado. En la recopilación bibliográfica llevada a cabo, que el principal mecanismo para abordar el desarrollo de interfaces de usuario en la actualidad está basado en la utilización del concepto de modelo. La denominación genérica para hacer referencia a las técnicas de desarrollo de interfaces de usuario es desarrollo de interfaces de usuario basado en modelos o Mb-UIDE. En las técnicas que dicho término aglutina no se ha alcanzado el nivel de estandarización que si que se ha logrado en otras disciplinas distintas a la Interacción Persona-Ordenador. En ella no hay lenguaje unificado de especificación ni tampoco un proceso unificado que guíe su desarrollo. Por ello está plenamente justificado el estudio, análisis y reflexión sobre las distintas propuestas realizadas hasta el momento.

Otro aspecto que también hemos identificado es la tendencia a estudiar y hacer propuestas en el sentido de la generación hacia adelante o forward de interfaces de usuario. Ese sentido es el predominante en las propuestas disponibles. Sin embargo, la identificación de métodos, procedimientos y la disponibilidad de herramientas para dar soporte al desarrollo inverso no está tan logrado ni ha recibido el mismo interés que sí que se constata en otras disciplinas, como en Ingeniería del Software.

Fruto de los estudios llevados a cabo hasta el momento y focalizando en el sentido inverso de desarrollo de interfaces de usuario, podemos concluir que, aunque existen algunos trabajos reseñables, éstos son pocos y tienen un alcance limitado, es decir no cubren todas las etapas

del cicdo de vida del sistema en el sentido inverso. También identificamos que las principales y más representativas propuestas Mb-UIDE no contemplan en detalle el sentido inverso a la hora de desarrollar interfaces de usuario. En este sentido, sólo la propuesta usiXML hace guiños significativos a ese sentido en el desarrollo. Sin embargo, identificamos, y nos apoyamos en las experiencias previas documentadas en Ingeniería del Software que el soporte a actividades de ingeniería inversa en general y de ingeniería inversa en interfaces de usuario conllevaría importantes beneficios relacionados con el mantenimiento y la portabilidad de productos software.

Las actividades de ingeniería inversa en interacción, aunque incipientes, pueden beneficiarse del caldo de cultivo que ha supuesto una importante actividad investigadora llevada a cabo por diferentes grupos de investigación a nivel europeo. En este sentido cabe mencionarse que en estos momentos existen distintas propuestas metodológicas que pueden ser adaptadas y estudiadas para dar soporte al sentido inverso y de lenguajes de especificación bastante extendidos y próximos a la estandarización o en vías de ello, como es el caso de usiXML.

En capítulos posteriores describiremos algunos pasos iniciales en el desarrollo de interfaces de usuario en sentido inverso y, más tarde, presentaremos una propuesta de tesis doctoral que pretende contribuir y definir un marco de trabajo que facilite al desarrollo inverso de interfaces de usuario.

Capítulo 3

Trabajo de Investigación

3.1 Introducción

En este capítulo se aborda uno de los trabajos de investigación ligado la propuesta de Tesis Doctoral que será descrita más tarde. Un primer problema ligado a la consideración conjunta del desarrollo de interfaces de usuario y de la ingeniería inversa consiste en la identificación de una herramienta que de soporte a la transformación entre modelos. Dicha actividad es básica si se pretende abordar la ingeniería inversa de manera automática o semi-automática. Por ello, a lo largo de este capítulo se identifican, presentan y analizan distintas notaciones que dan soporte a la transformación entre modelos.

El trabajo de investigación documentado en este capítulo sentará las bases para la realización de actividades de investigación posteriores, que serán identificadas en el próximo capítulo. El objetivo final es caracterizar de forma completa y sistemática el proceso de ingeniería inversa aplicado en contextos de interacción persona-ordenador. Pero para ello hay que determinar qué notaciones, modelos, métodos, herramientas y experiencia es necesario tener en cuenta.

En el trabajo de investigación descrito partiremos de dos premisas:

- El proceso de ingeniería inversa descansará en una propuesta de desarrollo basada/dirigida en modelos.
- Los principales modelos considerados serán los identificados en la propuesta Cameleon, descrita en el capítulo anterior.

A partir de esas dos premisas, en este capítulo, haremos un estudio inicial de lenguajes, notaciones, herramientas y desafíos que presenta el desarrollo inverso de interfaces de usuario basado en modelos.

Este capítulo estará divido en diferentes secciones. En primer lugar identificaremos las distintas transiciones a caracterizar, fruto de la consideración del contexto definido en Cameleon. Posteriormente, y utilizando algunos de los modelos en él propuestos identificaremos problemas y limitaciones para realizar el proceso de desarrollo inverso.

3.2 Ingeniería Inversa de Interfaces de usuario Basada en Modelos

Como ya lo hemos mencionado con anterioridad en el Capitulo 2 de esta memoria de fin de máster, una de las mayores ventajas y exigencias con las que se cuenta en el desarrollo basado en modelos de interfaces de usuario es la trazabilidad entre los mismos. En el proyecto Cameleon (proyecto europeo en el 5º programa marco) se identificaron una serie de modelos y transiciones que estarían presentes en el escenario asociado al desarrollo de interfaces de usuario tanto en el sentido forward como en el reverse.

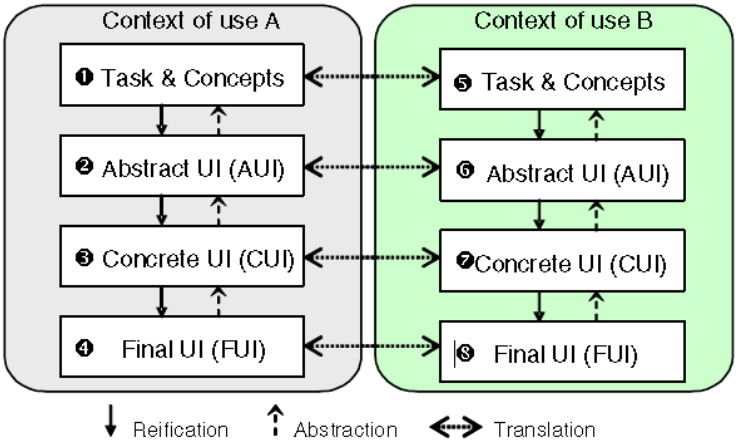

Ilustración 3-1 Marco simplificado definido en el proyecto Cameleon para dar soporte al desarrollo de IU

En la Ilustración 3-1 se identifican distintos modelos y transiciones definidas para dar soporte al desarrollo de interfaces de usuario basado en modelos. Los modelos que se identifican en la versión simplificada mostrada cubren tres aspectos: las tareas, los conceptos y la presentación (a distintos niveles de abstracción: abstracto, concreto y final). Las principales relaciones entre modelos son tres:

- Relaciones de reificación (o forward). Son las relaciones tradicionales que guardan los distintos modelos considerados cuando lo que se pretende es avanzar progresivamente hasta el producto final desde especificaciones independientes de la plataforma. Los modelos independientes de la plataforma son los asociados a la especificación de las tareas y de los conceptos. Los modelos de presentación, sin embargo, progresivamente van perdiendo nivel de independencia y llega un momento en que se alcanzan especificaciones concretas en un lenguaje de programación concreto. En ese momento el proceso de desarrollo habría alcanzado el nivel de interfaz de usuario final (Final UI).
- Relaciones de abstracción (o reverse). Son las relaciones que más nos interesa considerar en esta propuesta de tesis doctoral. Estas relaciones se definen también entre modelos, como las anteriores, pero en ellas se avanza en el sentido inverso al tradicional. Es decir, en las relaciones de abstracción definidos en Cameleon se parte de modelos de presentación, incluso de código, y se avanza en sentido inverso hasta alcanzar especificaciones independientes de la plataforma y de la modalidad, como las recogidas en los modelos de tareas y de conceptos. Más tarde identificaremos qué recursos hay disponibles para soportar la definición de relaciones de abstracción.
- Relaciones de traslación o traducción son aquellas que facilitarían la traducción de especificaciones realizadas bajo un contexto de uso a otros contextos de uso diferentes. Los contextos de uso se definen atendiendo al usuario, el entorno y la plataforma.

Fruto de la no haberse alcanzado una estandarización en las notaciones disponibles para la especificación de los distintos modelos considerados en la Ilustración 3-1 una de las primeras decisiones a tomar sería elegir aquella notación mejor situada para dar soporte a la realización de tareas de desarrollo inverso de interfaces de usuario.

Tal y como se estableció y justificó en el capítulo anterior, el lenguaje y la propuesta metodológica mejor situada para el fin perseguido es usiXML. Dicha notación está en proceso de estandarización por parte de la W3C y daría soporte a la especificación de modelos de tareas, dominio y presentación, pero no de definición de transformaciones modelo a modelo o modelo a texto. Por ello la siguiente actividad a realizar sería la revisión de notaciones y herramientas que permitan abordar la definición declarativa de transformaciones.

3.3 Lenguajes de especificación de transformaciones

Las transformaciones son la piedra angular del MB-UIDE ya que a partir de ellas se derivaran los nuevos modelos que permitirán la generación de las interfaces de usuario.

En (Kleppe, Warmer, & Bast, 2003) se definen las transformaciones como un conjunto de reglas de transformación que juntas describen como un modelo en el lenguaje fuente puede ser transformado en un modelo en el lenguaje destino.

En este apartado se describirán alguno de los lenguajes de transformación más usados en la transformación de modelos. Esas transformaciones son llevadas a cabo mediante el uso de herramientas que proporcionan un marco de trabajo para poder especificar las reglas de transformación. Entre las más destacadas podemos encontrar herramientas tales como XSLT, QTV, ATL; en esta sección se hará una descripción breve de cada una de ellas, enumerando algunas ventajas y desventajas que presentan cada una ellas.

3.3.1 Extensible Stylesheet Language Transformations (XSLT)

XSL Transformation (W3C, 1999), es un lenguaje declarativo que permite transformar documentos XML en otros documentos XML, documentos de texto o documentos HTML. XLST provee un conjunto de elementos que definen las reglas de como un documento XML debe ser transformado en otro documento XML. Las transformaciones pueden realizarse de forma dinámica.

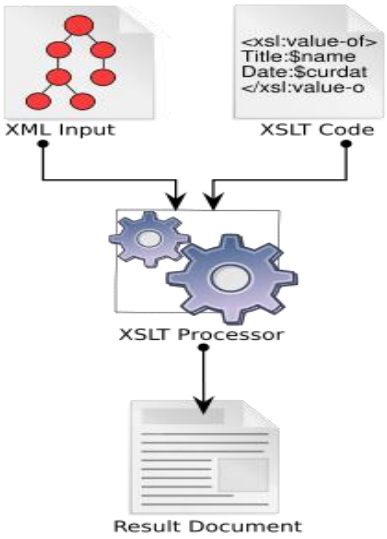

Ilustración 3-2 Modelo de Procesamiento XSLT

La Ilustración 3-2 el modelo de procesamiento de XSLT incluye los elementos siguientes:

- Uno o más documentos fuentes XML
- Una o más módulos de hojas de estilo XSLT
- Un procesador de plantillas XSLT
- Uno o más documentos de salida

En condiciones normales el procesador recibe dos documentos de entrada, un documento XML y una hoja de estilos XSLT que contiene un conjunto de reglas de plantillas, instrucciones y directivas que harán que el procesador produzca un documento de salida.

Todo documento XML que se encuentre construido correctamente tendrá una estructura de árbol, el procesador XLST con colaboración de XPath es el encargado que los documentos XML adquieran esa estructura, representando 7 tipos de nodos diferentes:

- The root
- Elements
- Text
- Attributes
- Namespaces
- Processing instructions
- Comments

Dicho de otra manera lo que hace realmente el procesador XSLT es convertir un árbol XML en otro árbol XML.

Es muy útil para poder dar formato a datos que se presentan por medio del navegador. Proporciona la facilidad de cambiar la apariencia o la estructura de un documento únicamente con cambiar la hoja de estilo XSL que contiene las reglas de transformación.

Algunas de las desventajas que se pueden encontrar a XSLT es la gran cantidad de código que se necesita escribir para poder realizar las transformaciones, lo que al final viene a desembocar en la dificultad de mantenimiento del mismo. También se presenta la necesidad de aprender un nuevo lenguaje, que puede significar mucho tiempo de aprendizaje dependiendo de la complejidad de las transformaciones que se vallan a realizar. Por otro lado al no ser un lenguaje de dominio de transformación de modelos no hace uso de los metamodelos, por lo que puede generar archivos de salida no validos.

3.3.2 *Query/View/Transformation (QVT)*

QVT(Query, View, Transformations) (OMG, 2005) es presentado como una especificación de la OMG, la especificación es presentada a lo largo de dos dimensiones ortogonales; la dimensión del lenguaje y la dimensión de la interoperabilidad.

Cada una de esas dos dimensiones se subdivide en varios niveles. En el caso de la dimensión del lenguaje define los diferentes tipos de lenguajes de transformación especificados en QVT. Específicamente son 3 los lenguajes de transformación: Relations, Core y Operational, la Ilustración 3-3 muestra la arquitectura bajo la cual se organiza la dimensión del lenguaje.

Su nombre se debe a los requisitos que debía cumplir el lenguaje según la OMG:

- **Q**uery (Consulta) son consultas que se realizan sobre los modelos que devuelven instancias de los tipos definidos en el lenguaje transformado.
- **V**iew (Vista) es un modelo obtenido en su totalidad a partir de un modelo base.
- **T**ransformations (Transformaciones) genera un modelo a partir de otro partiendo de un modelo origen, haciendo uso de metamodelos para la definición de las transformaciones.

En el nivel de lenguaje QVT es presentado como un hibrido que soporta lenguajes declarativos e imperativo, en la capa declarativa está formado por dos niveles de abstracción (véase Ilustración 3-3) formados por QVT Relations y QVT Core.

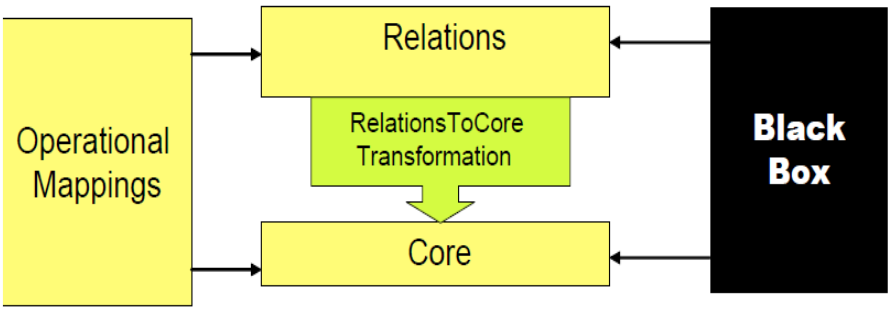

Ilustración 3-3 Arquitectura QVT

Fuente (OMG, 2005)

En QVT Relations las transformaciones son llevadas a cabo mediante la especificación de relaciones que deben mantenerse, entre los modelos candidatos, el lenguaje Relations permite la especificación de reglas declarativas. Puede soportar emparejamiento de patrones complejo, puede crear trazas implícitamente durante la transformación para poder observar que sucede durante el proceso. En este lenguaje la transformación entre dos modelos candidatos se expresa mediante un conjunto de relaciones que deben ser cumplidas para poder generar esta transformación. Un modelo candidato es un modelo que esta conforme a un tipo de modelo. En el siguiente ejemplo se muestra como la declaración llamada umlRdbms, el modelo uml y el modelo rdbms son modelos candidatos, en el caso del modelos uml este responde a las características del metamodelo SimpleUM y el modelos rdbms responde al metamodelo SimpleRDBMS.

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

}
```

Antes de realizar la transformación se comprueba la consistencia de las modelos candidatos con sus respectivos metamodelos. Otra característica con la que cuenta QVT Relations es que especifica condiciones que se deben cumplir para que una transformación pueda realizarse, una relación entre dos modelos es definida mediante dos o mas dominios y mediante las clausulas when y where que actúan para definir pre y post condiciones; un dominio es un tipo de variable que puede ser emparejada con un patrón en un modelo. Las relaciones pueden ser

de dos tipos, top relation y non-top relations, la diferencia entre ellas es que para que una transformación pueda llevarse a cabo todas las top relations deben estar resueltas.

El siguiente código define una transformación llamada umlRdbms, en ella se definen 3 relaciones, dos top relations que son PackageToSchema y ClassToTable y una non- top relation AttributeToColumn. Para que AttributeToColumn se pueda ejecutar tienen que cumplirse PackageToSchema y ClassToTable.

```
transformation umlRdbms (uml : SimpleUML, rdbms : SimpleRDBMS) {
    top relation PackageToSchema {…}
    top relation ClassToTable {…}
    relation AttributeToColumn {…}
}
```

Lenguaje Core presenta la misma potencia de Relation pero con una sintaxis mucho más simple , QVT Core que soporta coincidencia de patrones a través de un conjunto de variables mediante la evaluación de condiciones.

En caso de necesitar hace uso de lenguaje imperativo se hace uso de QVT Operational Mapping permite que se pueda realizar cualquier tipo de transformación expresada de manera imperativa, defiendo para ellos formas que permiten indicar los modelos involucrados en la transformación; también define una operación de entrada para su ejecución (llamada ***main***), al igual que las clases una transformación Operacional es una entidad instanciable con propiedades y métodos al igual que una clase. El código siguiente es un ejemplo de una regla de transformación imperativa.

```
transformation Uml2Rdbms(in uml:UML,out rdbms:RDBMS) {
// código para ejecución de las transformaciones
    main() {
       uml.objectsOfType(Package)->map packageToSchema();
           }
      ….
    }
```

Una de las ventajas que presenta QVT Operational es la capacidad de definir librerías que permiten la reutilización de código de las transformaciones, también permite la definición de "helpers" para definir operaciones sobre modelos y de la cual se necesita obtener un resultado, viene siendo el equivalente a un método en una clase.

Entre las ventajas que proporciona es que tiene variedad de lenguajes que permiten realizar las operaciones de acuerdo con las necesidades que se presenten.

Entre las desventajas que presenta QVT es que hay que conocer muy bien los meta-modelos involucrados en la transformación. Al indicar transformaciones de non- top relation, se involucran parámetros de transformaciones relations, lo que puede resultar un poco confuso.

Su curva de aprendizaje puede ser difícil debido a los diferentes lenguajes que engloba, debido a lo complejo que puede resultar su aprendizaje se han propuestos algunas

simplificaciones al lenguaje como la propuesta SimpleQVT(SQVT) presentada en (Giandini, 2007) y SmartQVT (Belaunde & Dupe) que es una implementación de código abierto realizada por France Telecom R&D.

3.3.3 *Atlas Transformation Language* (ATL)

Atlas Transformation Language (ATLAS group, 2005), es un lenguaje de transformación y un conjunto de herramientas (Edipse), que provee formas de obtener un nuevo modelo partiendo de un modelo fuente, ATL está basado en OCL(Object Contrainst Lenguaje) (OMG, 2006)

El lenguaje tiene la capacidad de poder definir tres tipos de unidades básicas:

- Módulos ATL
- Consultas ATL
- Librerías ATL

Cada uno de estas unidades puede estar compuesta por una combinaciones de "Helpers", atributos, y reglas.

Los modelos de origen y de destino deben estar en consonancia con sus respectivos metamodelos (véase Ilustración 3-4).

Al igual que QVT, ATL es un lenguaje hibrido que permite realizar transformaciones de forma declarativa o de manera imperativa según sea el caso. En ATL las transformaciones son unidireccionales, trabaja sobre modelos fuente de solo lectura y genera modelos de destino de solo escritura; durante la transformación el modelo fuente no acepta cambios sobre el; únicamente puede ser recorrido, mientras que el modelo destino puede aceptar cambios durante la transformación, pero no puede ser recorrido (navegado).

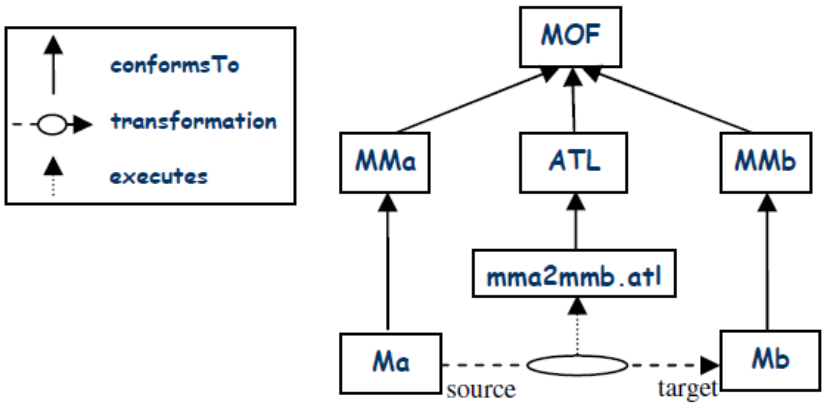

Ilustración 3-4 Enfoque de transformación de ATL

Fuente (Jouault & Kurtev, Transforming Models with ATL, 2005)

En (Jouault, Allilaire, Bézivin, & Kurtev, 2008) defiene las reglas de transformacion como la unidad basica sobre la cual se construye una transformacion en ATL, estas reglas de transformacion son llamadas "matched rules" , cada una de esas reglas esta compuesta de un patron origen y de un patron destino, estas reglas son ejecutadas cuando surge una

coincidencia en el patron origen. ATL permite la reutilizacion de codigo mediante la herencia de reglas, esta misma caracteristica permite el polimorfismo en la definicion de reglas.

ATL tiene una sintaxys muy parecida a la QVT, pero no hay interoperativadad entre ambos debido a las diferencias que presentan en sus metamodelos.

Retomando un poco los elementos que componente ATL se comenzara a describir los modulos, ya que representan una de las unidades mas importantes que componen ATL.

Los módulos representan la unidad básica de transformación, es el programa en el que se especifican las reglas que darán origen a una transformación modelo a modelo.

Un modulo se define mediante la sentencia “module”, que indica que se tratara de una transformación de modelo a modelo, y la sentencia “créate” que indica cual será el modelo de destino y el modelo de origen.

```
module C2A;
create OUT : MM from IN : MM;
```

Los “helpers” pueden verse como un equivalente a los métodos que forman para de las clases. Permite la especificación de código ATL que puede ser reutilizado, este código puede ser llamado desde cualquier punto del modulo.

Los helpers se definen mediante la sentencia ”helper”, pueden ser definidos como tipo OCL o un tipo relacionado con el metamodelo de origen. En el siguiente ejemplo se define un helper para almacenar en la variable auiModel, el elemento raíz del modelo; es decir el modelo que contendrá a los componentes abstractos de interacción y a los contenedores de interacción abstractos.

```
helper def: uiModel : MM!uiModel = MM!uiModel.allInstances()->first();
```

Como se menciono anteriormente un programa ATL está conformado por reglas que definen como los elementos de un modelo de origen se corresponde con los elementos de un modelo de destino.

Las reglas se podrían describir como la unidad básica con la cuenta ATL para poder llevar a cabo la transformación entre los modelos. Como ya se había mencionado antes en este documento, estas reglas pueden ser especificadas de una forma declarativa o bien imperativamente, inclusive haciendo una combinación de ambas.

La parte declarativa de ATL está constituida por las Matched Rules, estas reglas están destinadas a encontrar patrones dentro del metamodelo de origen para generar elementos de salida que correspondan al metamodelo destino, por lo consiguiente estas reglas están conformadas por un patrón origen y un patrón destino

El ejemplo siguiente describe la especificación de una regla declarativa, como se puede observar la definición de la misma comienza con la sentencia “rule” seguida por el nombre que

de regla. La sentencia "from" indica cual será el patrón fuente y la sentencia "to" el patrón de destino. En muchas ocasiones es necesario indicar código imperativo durante las transformaciones, en ATL esto es posible de realizar dentro de las reglas declarativas haciendo uso de la sentencia do para especificar código imperativo.

```
rule  createRootAUIContainer {
  from
   s : MM!CioType(s.refImmediateComposite().oclIsKindOf(MM!cuiModel))
  to
   t : MM!abstractContainer(id<-s.id, name<-s.name)
  do {
      thisModule.uiModel.auiModel.abstractContainer<-t;
     }
}
```

A parte de las reglas declarativas ATL permite la definición de reglas que son llamadas por otras reglas; se denominan "called rules", generalmente estas reglas reciben parámetros de entrada para ejecutar un procedimiento.

El siguiente ejemplo muestra la forma como se define una "called rule" donde se definen dos parámetros de entrada, para la ejecución del código imperativo definido en ella.

```
rule setModelAttributes(s: MM!uiModel, t: MM!uiModel) {
      do {
            t.id   <- s.id;
            t.name <- s.name;
      }
}
```

Unca caracteristica que posee ATL es que permite herencia en la definicion de las reglas, lo que permite reutilizacionde codigo.

Entre las ventajas que suponen el uso de ATL es la naturaleza hibrida del lenguaje que permite la definicion de reglas declarativas e hibridas, con la la herencia de reglas permite la reutilizacion de codigo y el polimorfismo de reglas, existen herramientas completas que dan soporte al lenguaje como en el caso de Eclipse.

Entre las deventajas encontradas aparecen la necesidad de aprender la sintaxis de un nuevo lenguaje, una vez que se inicia el proceso de transformacion el modelo origen no puede ser modificado en caso de ser necesario, y el modelo destino no puede ser navegado.

3.3.4 Transformación de Grafos

La gramática de grafos puede resultar de mucha utilidad para poder transformar modelos, debido a que un modelo puede ser representado mediante un grafo, donde las clases representan nodos conectados mediante asociaciones que representan las aristas.

La transformación por grafos ha sido muy utilizada para la transformación de modelos, especialmente en la transformación de modelos visuales. En (Taentzer, et al., 2005) se

menciona que los problemas de transformación de modelos pueden ser descritos como un problema de transformación de grafos (véase Ilustración 3-6).

(Ehrig, Herig, & Prange, 2006) menciona que la principal idea de la transformacion por grafos es la modificacion de grafos basada e reglas (vease Ilustración 3-5).

El corazon de una regla de transformacion es un par de grafos, en el caso de la Ilustración 3-5, la regla estaria definida por "***p***" y el par de grafos serian "***L***" como origen y "***R***" como destino.

"L" represente un subconjunto del modelo origen a transformar y "R" contiene el patron de destino, cuando se aplica la regla "p" sobre ellos entonces "L" es sustituida por "R".

Cuando se aplica la regla "***p***" sobre los nodos L y R, lo que se busca es una coincidencia de L en el grafo origen y se reemplaza por R.

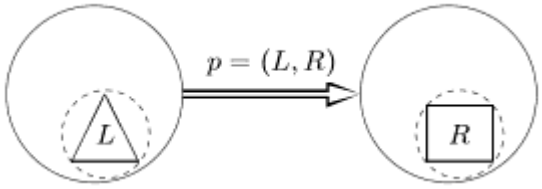

Ilustración 3-5 Modificación de Grafos Basada en Reglas

fuente (Ehrig, Herig, & Prange, 2006)

Al usar el enfoque de transformación por grafos tanto el modelo fuente "L" como el modelo destino "R" tienen que ser proporcionados en forma de grafos. Al realizar la transformación entonces la regla hace uso de la sintaxis abstracta del modelo de entrada y obtiene la sintaxis abstracta del modelo de salida. La sintaxis abstracta del grafo de entrada poder ser un subconjunto de la instancia del grafo inicial. Por ejemplo tomando en consideración la Ilustración 3-6 se puede observar que el grafo que describe la interfaz de usuario esta formado por los nodos "Windows","button" y "label", al especificar la regla de transformación de la ventana en una transformación por grafos entonces se le proporciona como entrada el grafo correspondiente a la ventana que contiene la sintaxis abstracta únicamente de ese elemento , pero si se quiere realizar la transformación de un "button" o un "label" entonces tenemos que proporcionarle el modelo fuente correspondiente al subconjunto "button" o "label" ya que ambas son instancias del grafo principal que describe a la ventana, en este punto la regla de transformación no tiene en consideración la sintaxis abstracta del nodo "window" ya que pertenece a otro subconjunto. .

La Ilustración 3-6 muestra la forma como se representa una interfaz de usuario usando grafos y atributos.

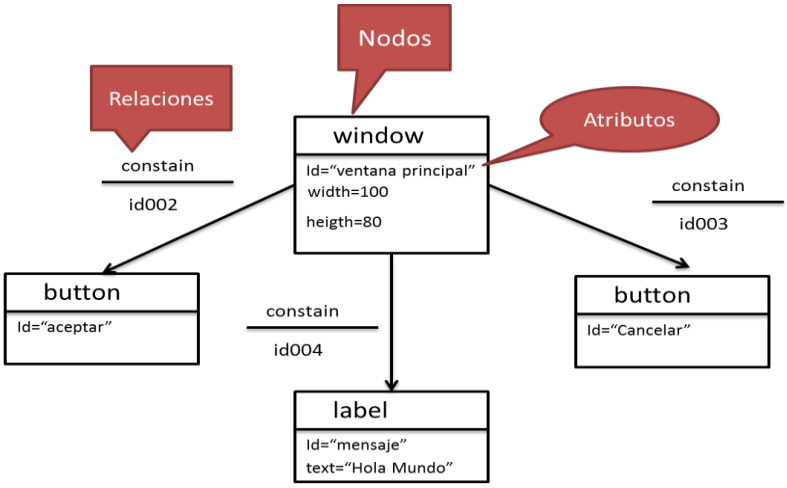

Ilustración 3-6 Representación de una Interfaz de usuario mediante un Grafo

El principal problema de esta tecnica consiste en buscar la forma de como borrar "***L***" una vez que la regla ha sido ejecutada, y poder conectar "***R***" en el conntexto del grafo destino.

Otro problema que puede presentar es lo complejo que puede resultar su manejo cuando una interfaz de usuario cuenta con muchos componentes, teniendo en cuenta que cada componente seria un nodo al igual que cada uno de atributos.

Como ventaja se puede mencionar que ya ha sido utilizado en la transformacion de interfaces de usuario, especificamente en UsiXML, lo que garancitiza su funciomiento en el MB-UIDE, para mayores referencias al respecto consultar (Limbourg Q. , Multi-Path Development of User Interfaces, 2004) y (Limbourg Q. , Vanderdonckt, Michotte, Bouillon, & López-Jaquero, 2005).

Esta transformación esta soportada por herramientas como AGG (AGG Homepage), AToM (Lara & Vangheluwe, 2002), VIATRA2 (Varro & Pataricza, 2004) y VMTS (Levendovszky, Lengyel, Mezei, & Charaf, 2004)

3.3.5 Comparativa y conclusiones

Lenguaje de Transformación	Características	Ventajas	Desventajas
XLST	Convierte archivos XML en cualquier tipo de archivos. Necesita un procesador que aplique las reglas y realice la conversión de archivos. La hoja de estilos también es un archivo XML, hace uso de XPATH para formar la estructura de los documentos.	Únicamente se necesita realizar cambios en la hoja de estilos XSL cuando se necesitan realizar modificación en los documentos de salida. El formato del documento de salida puede ser el que especifique el usuario, no necesariamente tiene que ser un documento en formato XML.	Hay que escribir mucho código para una transformación pequeña. Debido a la gran cantidad de código que hay que escribir el mantenimiento del mismo puede ser una tarea compleja. Alto consumo de memoria al momento de realizar transformaciones, esto debido a los arboles generados por XPAHT con la estructura del documento de entrada. No hace uso de metamodelos por lo que los modelos generados de una transformación pueden ser inválidos.
QVT	Permite definir transformaciones entre metamodelos definidos bajo MOF. Permite flexibilidad en la implementación. Permite la creación de la vista de un modelo.	Tiene variedad de lenguajes para poder realizar las transformaciones, esta soportado por EMF, lo que le proporciona estabilidad. Tiene Naturaleza Hibrida; declarativa e imperativa.	Se necesita conocer los metamodelos correctamente para poder definir la transformación. Se presenta la necesidad de aprender un nuevo o varios lenguajes
ATL	Es un lenguaje de transformación de modelos y un conjunto de herramientas. Los modelos fuente y destino corresponden a la definición de sus respectivos metamodelos.	Está basado en MOF por lo que esta soportado por Edipse Modeling Framework, hace uso de OCL lo que da flexibilidad al consultar los modelos. Permite la definición de reglas imperativas y declarativas e inclusive una combinación de ambas en caso de necesitarse.	En caso de transformaciones complejas es posible tener que recurrir a la sintaxis de OCL, por lo que en casos como este la curva de aprendizaje es más larga.
Transformación por Grafos	Expresa los modelos en términos de grafos, atributos.		La eliminación del patrón fuente puede ser un problema si no se definen bien las reglas. Se necesita conocimiento preciso del metamodelo.

3.4 Caso de estudio: transformación de CUI a AUI

En este punto se abordara un caso de estudio en el cual se realiza transformaciones de abstracción entre un modelo de interfaz concreta y un modelo de interfaz abstracta, usando como metodología y como lenguaje de transformación UsiXML.

La Ilustración 3-7 corresponde a una ventana de impresión, como se puede observar es una ventana rica en objetos de interacción concreta. Se pueden identificar Listas desplegables, botones de comando, hipervínculos, botones de radio, cuadros de texto, spinner, trackbar, ventanas, etiquetas, imágenes decorativas, casillas de verificación y un visor de vista previa del documento a imprimir.

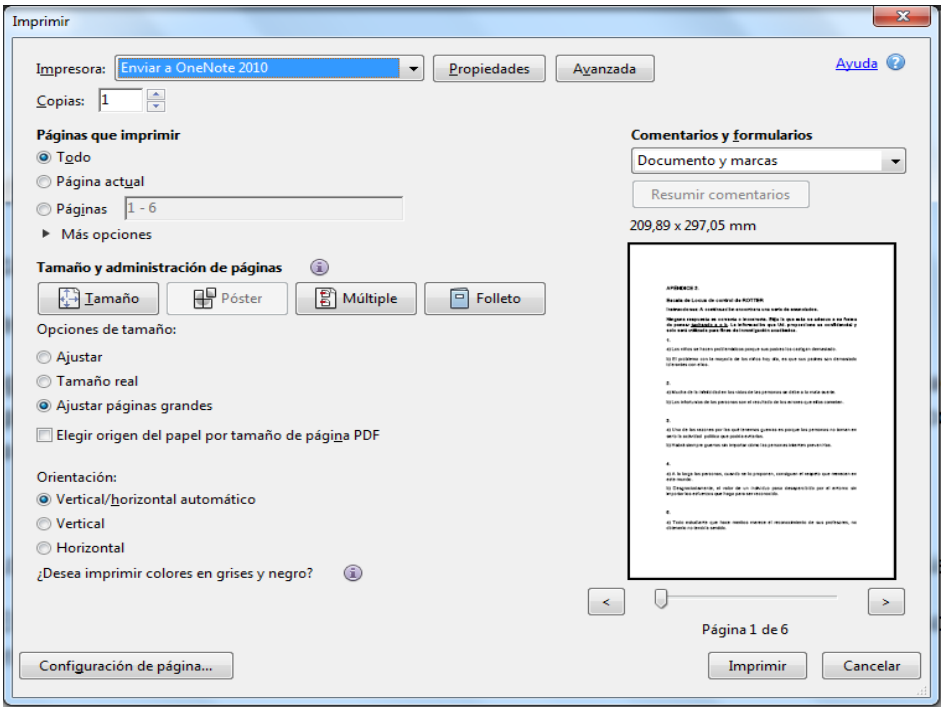

Ilustración 3-7 Pantalla de Impresión

Para poder plantear una transformación es necesario establecer las relaciones entre los elementos del modelo de interfaz concreta y el modelo de interfaz abstracta. En el siguiente apartado se definirán las relaciones entre el modelo origen y el modelo destino.

3.4.1 Elementos de origen y destino de la transformación

Es importante tener en cuenta que una de las principales ventajas con las que cuenta UsiXML en el modelo de interfaz de usuario abstracta, es que se expresa en términos de componentes de interacción abstracta, donde existen contenedores abstractos (abstractContainers) y componentes individuales de interacción, estos últimos toman una faceta dependiendo del comportamiento que vallan a tener. Esta faceta puede ser "input" en el caso que el componente individual de interacción reciba entrada de datos por parte del usuario, la faceta "output" representara aquellos componentes individuales de interacción que muestran salida de datos, los elementos que proporcionan ayuda en la navegación de la interfaz tendrán una faceta "navigation" y los elementos que desencadenan la ejecución de una tarea tendrán una faceta "control".

Teniendo en cuenta lo anterior entonces se procede a identificar los elementos que se encuentran en la interfaz de usuario concreta y de acuerdo a la función que desempeñan entonces relacionarlo con el componente de interacción abstracta que corresponda.

Interfaz de Usuario Concreta	UsiXML	Metamodelo UsiXML	Interfaz de Usuario Abstracta
Ventana	window	windowType	AbstracContainer
Botones	button	ButtonType	AbstractindividualComponent Faceta: control
Cuadros de texto	inputText	InputTextType	AbstractindividualComponent Faceta: Input
Hipervínculos	outputText	OutputTextType	AbstractindividualComponent Navigation
Botones de Radio	radioButton	radioButtontype	AbstractindividualComponent Faceta: Input
Listas desplegables	comboBox	ComboBoxType	AbstractindividualComponent Faceta:Input
Trackbar	Slider	SliderType	AbstractindividualComponent Faceta: Output
Casillas de Verificación	checkBox	CheckBoxType	AbstractindividualComponent Faceta: input
Spinner	Spin	SpinType	AbstractindividualComponent Faceta: Input
Panel de Pre visualización de Documento	-	-	AbstractindividualComponent Faceta: output
Etiquetas	outputText	OutputTextType	AbstractindividualComponent Faceta: output
Imágenes (decorativas)	imageComponent	ImageType	AbstractindividualComponent Faceta: output

Tabla 3.1 Relación entre componentes de interacción Concreta y Abstracta

La Tabla 3.1 muestra la relación existente entre los elementos de la interfaz de usuario concreta y su equivalente en la interfaz de usuario abstracta. En este caso de estudio tenemos un abstractContainer en el que estarán contenidos el resto de los elementos.

3.4.2 Definición de las transformaciones

Una forma de poder entender cómo se realizaran las transformaciones es definiendo un algoritmo para la misma, debemos tomar en cuenta que los programas que contienen las reglas de transformación en Lenguajes como ATL y QVT no siguen una ejecución secuencial, al ser declarativos pueden ejecutar las reglas en el orden en el que se encuentren los patrones en el modelo de entrada, pero definir un algoritmo puede servir como una guía a seguir sobre el comportamiento que debe tener la transformación al momento de asignar "padres" a los elementos.

Inicio
Recorrer todos los CUI que pertenecen a la interfaz
    Para cada ventana en la CUI
        Crear un contenedor abstract
        Para cada control que se encuentre en la CUI
            Crear un AbstractComponent
            Seleccionar faceta del abstractcomponent
                Case **"**permite entrada de datos**"**
                    Crear faceta input en el abstractcomponent actual
                Case "el componente muestra información de salida"
                    Crear faceta output al abstractcomponent actual
                Case "el componente proporciona ordenes al ordenador"
                    Crear faceta de control en el abstractcomponent actual
                Case "el componente permite desplazamiento"
                    Crear faceta de navegación
            Fin Seleccionar
        Fin Para
    Fin Recorrer
Final

El algoritmo anterior recorre todos los componentes de la interfaz cuando se encuentre un contenedor de componentes gráficos(ventana) como una ventana entonces se encarga de crear un abstractContainer, por cada control que se encuentre dentro del contenedor de componentes gráficos(ventana) entonces se procede a la creación de un componente abstracto (abstractComponent), de acuerdo a la naturaleza de la función que desempeñe ese componente entonces se le asigna una faceta, que puede ser input, output, control y navigation.

Las transformaciones serán realizadas usando el metamodelos UsiXML.ecore por lo que es necesario definir un lenguaje de transformación que permita hacer uso del mismo. En el apartado anterior pudimos observar que hay dos lenguajes que cumplen con ese requisito. ATL y QVT, ambos esta basados en MOF, por lo que están soportados por Edipse Modeling Framework. En este caso se hará uso de ATL como lenguaje de transformación, ya que es muy fácil de aprender con respecto a QVT. Antes de poder llevar a cabo la transformación es necesario poder tener el modelo de entrada que corresponde al modelo de interfaz de usuario concreta, como se está partiendo de una impresión de pantalla no se dispone del modelo de entrada por lo que es necesario generarlo. Se usara el editor GrafiXML (Michotte & Vanderdonckt, 2008) en cual provee la posibilidad de diseñar una interfaz de usuario y obtener

su especificación en XML conforme al metamodelo de UsiXML. Esta es una forma "manual" de obtener el código XML de la interfaz ya que no existe una herramienta que permita obtener de una captura de pantalla el código XML correspondiente a dicha interfaz y que responda al metamodelo de UsiXML, por lo que se podría plantear la posibilidad de extender GrafiXML para que pueda realizar esta tarea o bien el desarrollo de una nueva herramienta que de soporte al proceso de ingeniería inversa en este sentido.

Retomando la Ilustración 3-4 podemos modificarla para representar el esquema de transformación en el ámbito de este trabajo.

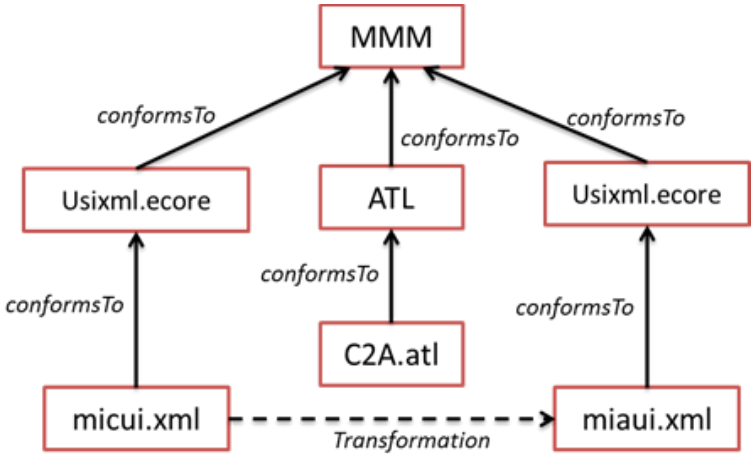

Ilustración 3-8 Enfoque de Transformación Modificado

La Ilustración 3-8 reprenta la forma como estara formado el esquema de transformacion para aboradar este caso de estudio, MMM representa al metamodelo proporcionado por Metaobject Facilities, el metamodelos de entrada y el metamodelo de de salida sera el metamodelo de UsiXML, ya que se parte del modelo de interfaz de usaurio concreta definida en UsiXMl y se obtendra por medio de la transformacion el modelo de interfaz de usuario abstracta tambien de UsiXML por ello usamos el mismo metamodelo como origen y destino, el lenguaje de transformacion sera ATL por lo que definiremos un modulo de transformacion llamado "*C2A.atl*" que contendra las reglas de transformacion que haran posible la transformacion. Como mencionamos anteriormente es necesario tener la espcificacion en XML de la interfaz conforme al metamodelo UsiXML, esa especificacion la guardaremos en un archivo llamado micui.xml, el cual sera la el modelo origen de la transformacion, miaui.xml representa al modelo de salida que obtedremos de la transformacion.

En vista que hemos elegido ATL para la transformacion es necesario definir los siguientes elementos:

- Modulos
- Helpers
- Reglas
- Consultas

Los modulos son la unidad basica de transformacion, es el programa en el que se definen las reglas, helpers y consultas. El modulo debe contener un encabezado el cual tiene la informacion acerca de los metamodelos de entrada y salida, se define de la siguiente forma.

```
module C2A;
create OUT : MM from IN : MM;
```

La palabra *module* define el inicio del modulo debe ir seguido por el nombre del mismo, el nombre que se le da al modulo debe corresponder con el nombre del fichero que se ha concreado con el fin de definir las reglas de transformación. También debe contener la sentencia créate la cual define los metamodelos fuente y destino, en el ejemplo anterior lo que se planeta es que a partir(from) del metamodelo MM cree una salida(out) en este caso como se está transformando de código UsiXML a UsiXML el metamodelo de entrada y salida es el mismo.

Lo siguiente será definir las reglas de transformación llevaran a cabo la transformación entre los modelos, es necesario tener en consideración antes de iniciar a escribir las reglas que elementos transformaremos, para ellos es necesario tener claro como esta estructurado el modelo de interfaz concreta de UsiXML (véase Ilustración 3-9).

```
<uiModel>
 <cuiModel>
    ….Elementos de Interfaz Concreta
 </cuiModel>
</uiModel>
```

Ilustración 3-9 Esquema CUI Model

La Ilustración 3-9 indica que el modelo de interfaz de usuario concreta esta contenido dentro del modelo de interfaz de usuario (uiModel), de igual forma el modelo de interfaz de usuario abstracta (auiModel) está contenido dentro del modelo de interfaz de usuario(uiModel) solo que en diferentes niveles de abstracción(véase Ilustración 3-10).

```
<uiModel>
 <auiModel>
    ….Elementos de Interfaz Concreta
 </auiModel>
</uiModel>
```

Ilustración 3-10 Esquema auiModel

Al pertenecer ambos al modelo de interfaz de usuario (uiModel) en diferentes niveles de abstracción y por consiguiente en diferentes modelos se tiene que escribir una regla de transformación que tome como elemento de fuente el modelo de interfaz de usuario

(uiModel) y como destino el modelo de interfaz de usuario (uiModel) pero dentro del ámbito de la interfaz de usuario abstracta.

```
rule Init {
    from
        s: MM!uiModel
    to
        t: MM!uiModel ( auiModel<-s.auiModel),
        t1: MM!auiModel()
    do {
        thisModule.uiModel <- t;
        thisModule.setModelAttributes(s,t);
        t.schemaVersion <- s.schemaVersion;
        t.creationDate <- s.creationDate;
        t1.id <- 'auikModel_01';
        t1.name <- 'CUI2AUI 01';
        t.auiModel <- t1;
        t.auiModel  <- MM!cuiModel.allInstances()->select(e | e.refImmediateComposite().oclIsTypeOf(MM!cuiModel))->first();
        }
}
```

Ilustración 3-11 Regla de Transformación del Modelo de Interfaz de Usuario (uiModel)

La Ilustración 3-11 describe la definición de la regla para transformar el modelo de interfaz de usuario de concreta a abstracta, la palabra "*rule*" indica que se está definiendo una regla de transformación declarativa, en el caso anterior "Init" indica el nombre que se ha asignado a la misma, las instrucciones que llevaran a cabo la transformación se escriben entre los corchetes. El elemento "from" indica cual es el elemento fuente, y "to" establece cual será el elemento destino. En este caso el elemento fuente y destino es el mismo (MM!uiMOdel), una regla puede tener varios elementos destino como en el caso anterior, podemos observar que el primer elemento destino es asignado a la variable "t" y el segundo elemento destino es asignado a la variable "t1". Esta regla también contiene código imperativo el cual está definido dentro del segmento de código delimitado por la palabra "do", en esta sección lo que se está indicando es que a los elementos destino se les asigna las información relacionada con los datos de creación, el nombre y el id del modelo.

Una vez creado los modelos "uiModel" y "auiModel" entonces ya se puede proceder a la definición de las reglas que permitirán la transformación de los componentes de interacción concreta en componentes de interacción abstracta.

Para especificar las reglas de transformación para los componentes de interacción debemos aclarar que la nomenclatura a usar en los elementos origen correspondientes al modelo de interfaz de usuario concreta, será la que se definió en la segunda columna de la Tabla 3.1, ya que es la forma como están expresados los componentes de interacción concreta en el metamodelo de UsiXML. También es necesario aclarar que alguno de los componentes de interacción concreta que usa la pantalla que se está tratando en este caso de estudio están disponibles en el metamodelo; por ejemplo el componente de pre visualización dé la impresión y el hipervínculo de ayuda. Por lo que para efectos de este trabajo el componente de pre visualización será tratado como una imagen, esto es debido a que el componente de vista previa lo que hace es mostrar información al usuario por lo que se convierte en un componente individual de interacción con una faceta output al igual que la imagen, el caso de hipervínculo se tratara como una etiqueta, ya que presentan características similares, la diferencia seria en que al ser transformado.

El primer componente de interacción concreta que se debe tratar una vez que se han transformado los modelos de interfaz de usuario (uiModel) y el modelo de interfaz de usuario concreta (cuiModel) al modelo de interfaz de usuario abstracta (auiModel) son lo componentes que actúan como contenedores; en este caso la ventana, una ventana es un contenedor ya que dentro de ella se encuentran otros componentes de interacción.

Para definir la regla de transformación de la ventana debemos referirnos a la Tabla 3.1 donde podemos observar que el componente correspondiente al metamodelo de UsiXML relacionado con la ventana (Windows) es el WindowType, por lo que WindowType será el elemento fuente en la regla de transformación, también se puede observar en la misma tabla que el componente de interacción abstracta correspondiente al elemento de fuente "WindowType" será una "AbstractContainer" véase(Ilustración 3-12).

```
rule createRootAUIContainer{
      from
        s : MM!CioType(s.refImmediateComposite().oclIsKindOf(MM!cuiModel))
      to
        t : MM!abstractContainer(id<-s.id, name<-s.name)
      do {
        thisModule.uiModel.auiModel.abstractContainer<-t;
        }
}
```

Ilustración 3-12 Regla de Transformación Root Abstract Container

Una vez transformado el elemento Windows en una abstractContainer entonces se procede a la transformación de los componentes que representan la disposición de los elementos dentro del contenedor (Windows), este componente es denominado Layout, el componente Layout puede ser de 4 tipos diferentes: FlowBox, GridBox, BorderBox, GridBagBox, todos están contenidos dentro de la ventana y son mutuamente excluyentes por lo que una ventana únicamente puede tener uno de ellos, la diferencia entre cada uno de ellos es la forma como los componentes de interacción concreta están distribuidos dentro de la ventana.

La siguiente regla de transformación toma como fuente un Layout de tipo GridBox y lo transforma en un abstractContainer, este nuevo abstractConiner que se define con esta regla es asignado al abstractContainer correspondiente a la ventana.

```
rule createAUIContainer{
    from
      s : MM!CioType(s.oclType().toString()='MM!GridBoxType')
    to
      t : MM!abstractContainer(id<-s.id, name<-s.name)
    do {
      MM!abstractContainer.allInstances()->select(e | e.id = s.refImmediateComposite().id)->first().abstractContainer <- t;
      }
}
```

Ilustración 3-13 Regla de Transformación AbstractContainer

El Layout contiene a los elementos individuales de interacción en la interfaz de usuario concreta, por lo que en la interfaz de usuario abstracta tendrá la misma función, todos los

componentes individuales de interacción que se creen a partir de las siguientes transformaciones que se definan serán agregados al abstractContainer que corresponde al GridBox (véase Ilustración 3-13).

En la siguiente regla se definirá la transformación de la etiquetas, si regresamos a la tabla 3.1 se puede ver que las etiquetas en UsiXML se representan mediante outputText y el tipo de elemento en el metamodelo es OutputTextType que se corresponde con un abstractIndividualComponent con una faceta output. Al definir la regla de transformación tendremos como elemento fuente a OutputTextType y como elementos destino a abstractIndividualComponent y a output como faceta (véase Ilustración 3-14).

```
rule createOutputIndividualAbstractComponents {
    from
      s:MM!CioType(s.oclType().toString()='MM!OutputTextType')
    to
      t: MM!abstractIndividualComponent(id <- s.id,name <- s.name ),
      i: MM!output(id <- 'output_' + s.id, name <- 'output_' + s.name)
    do {
      t.output <- i;
     MM!abstractContainer.allInstances()->select(e | e.id = s.refImmediateComposite().id)->first().abstractIndividualComponent <- t;
      }
}
```

Ilustración 3-14Regla de Transformación para crear Componentes Individuales de Interacción

De la misma forma que en la que definió la regla anterior se pueden definir el resto de las reglas relacionadas con los componentes individuales de interacción, la única diferencia sería en el tipo de componente de interacción que servirá como entrada y la faceta que tendrá la salida (véase Tabla 3.1), por ejemplo si se quiere de transformar un cuadro de texto, en el metamodelo de UxiXML es un InputTextType por lo que el componente de interacción concreta que actuara como fuente seria `MM!InputTextType` y la faceta seria output.

De esta forma describimos las reglas de transformación que permiten las relaciones de abstracción entre los modelos de interfaz de usuario concreta e interfaz de usuario abstracta.

3.4.3 Análisis del proceso de transformación

El caso de estudio descrito anteriormente demuestra que usando UsiXML es posible realizar relaciones de abstracción entre los modelos de interfaz de usuario concreta e interfaz de usuario abstracta.

El resultado de transformación es un documento XML conforme a la especificación UsiXML el cual está expresado en términos de contenedores abstractos y componentes individuales de interacción.

Es necesario mencionar que conforme pasa el tiempo aparecen más componentes de interacción que son usados en las interfaces de usuario, esto debido a la necesidad de brindar mayores facilidades al usuario. Muchos de esos componentes aun no están considerados en las metodologías aplicadas al MB-UIDE, lo que podría representar un problema al querer aplicar transformaciones de abstracción de esos componentes. Uno de los problemas que se encontraron en esta transformación está relacionada con lo planteado anteriormente, por

ejemplo si tomamos en consideración el caso del componente concreto de vista previa del documento veremos que UsiXML no tiene un componente considerado para ese elemento, por consiguiente los editores como GrafiXML cuya función es modelar una interfaz de usuario concreta no consideran tales elementos en sus barras de herramientas lo que obliga a usar otro componente de interacción que presente características similares.

La interfaz de usuario propuesta para este caso de estudio cuenta con mas de 40 componentes de interacción concreta, y en un lenguaje como ATL bastaron 12 reglas de transformación (incluida la transformación de modelos) para poder especificar relaciones de abstracción entre los modelos. Al ser ATL un lenguaje especifico del dominio de transformación de modelos permite generar modelos validos como salida y tener una visión muy ingenieril de la transformación de modelos, además se cuenta con la ventaja que el código se puede reutilizar.

La documentación completa de este caso de estudio puede verse en el Disco Compacto que acompaña a este trabajo de investigación.

3.5 Conclusiones y trabajo futuro

En este capítulo se presentaron alguno de los lenguajes de transformación que dan soporte tanto al MDE como al MB-UIDE y de las ventajas que supone su uso. Se presentaron lenguajes como XSLT que dan soporte a la transformación de modelos debido a que las instancias de los mismos esta descritos en ficheros XML y XSLT está concebido para la transformación de archivos XML en otros archivos XML o bien en cualquier otro tipo de documento, la principal desventaja que presenta XSLT es que no hace uso de los metamodelos por lo que es posible que generen archivos no validos como modelos de destino, además que en transformaciones complejas el código se puede volver un poco inmanejable y difícil de mantener en caso que no es estructure y documente bien.

En el caso de ATL y QVT ambos presentan características similares, ambos están conforme a MOF (Meta Object Facilities), por lo que hacen uso de metamodelos para realizar transformaciones, lo que permite validar tanto los modelos de entrada como de salida evitando de esta manera que se produzcan archivos inválidos.

El caso de estudio planteado en este capítulo ha demostrado que es posible lograr transformaciones de abstracción entre los modelos de interfaz de usuario concreta y la interfaz de usuario abstracta, usando la propuesta metodológica UsiXML y como lenguaje de transformación.

UsiXML es un lenguaje y una propuesta metodológica que es rica en componentes concretos de interacción aun así en su versión 1.8 es posible observar que hay componentes que no están considerados dentro de su metamodelo, por consiguiente los editores de interfaces de usuario concreta que dan soporte a UsiXML tampoco cuentan con esos Widget en sus barrar de herramientas, el claro ejemplo de ello es el visor de vista previa de documentos, habrá que esperar un poco la versión 2.0 para considerar el desarrollado de nuevas herramientas como editores que den soporte al proceso de ingeniería inversa de interfaces de usuario basadas en modelos haciendo uso de UsiXML como propuesta metodológica.

Lo más importante de lograr las relaciones de abstracción por medio de transformaciones es que se puede volver a generar la interfaz de usuario, permitiendo realizar migraciones, o cambios de plataforma, haciendo uso de las herramientas que ya existen y dan soporte a esa tarea, otra ventaja es que se logra re-documentar la interfaz, por lo que de una misma tarea se obtienen varios logros representativos.

Capítulo 4

Anteproyecto de Tesis

4.1 Introducción

Este capítulo está orientado a identificación y propuesta de un anteproyecto de tesis doctoral. En este sentido, se identificarán, a lo largo del presente capítulo, distintas actividades y tareas relacionadas con el proceso inverso de generación de interfaces de usuario. Es decir, aquel proceso que parte de especificaciones concretas de interfaces de usuario y permite alcanzar distintos modelos asociados a las tareas y dominio, es decir, permite abordar la obtención de especificaciones independientes de la plataforma.

4.2 Propuesta de Tesis

La propuesta de la tesis doctoral esta orienta a la obtención de niveles de abstracción más altos asociados a la especificación de interfaces de usuario, partiendo de otros modelos más concretos. Es decir; nuestra propuesta se centra en el proceso inverso de generación de interfaces de usuario, que habitualmente han considerado las distintas propuestas disponibles en la literatura y ligadas al Mb-UIDE. Nuestra propuesta de tesis doctoral resulta interesante y beneficiosa desde distintos puntos de vista:

- Permite dar soporte a la migración de interfaces de usuario elaboradas para una plataforma concreta y su paso a otras plataformas

  Con los avances tecnológicos han surgido una gran cantidad de dispositivos en el mercado, cada uno de ellos con características propias, por lo que el desarrollo de interfaces de usuario varía de un dispositivo a otro, todos utilizan sus propias API's para el desarrollo de aplicaciones por los que existes diferencias considerables entre los widgets que usan para la especificación de interfaces de usuario. Con el enfoque basado en modelos en los niveles de abstracción más altos se alcanza independencia de la plataforma, lo que beneficia el poder generar interfaces de usuario tanto para dispositivos móviles, interfaces web, y de escritorio partiendo de la misma especificación.

  Es por ello que si se logra lograr la abstracción (ingeniería inversa) de una interfaz de usuario mediante el enfoque basado en modelos entonces se podría regenerar esa interfaz para cualquier plataforma sin necesidad de redefinirla de nuevo.

- Ofrece soporte a la modificación/adaptación/mantenimiento de interfaces de usuario

  En muchas organizaciones es muy común encontrar sistemas cuya antigüedad se remonta a muchos años y de los cuales no se cuenta ya con soporte por parte del fabricante. Los problemas comienzan a surgir cuando el sistema deja de satisfacer las necesidades de los usuarios y estos comienzan a solicitar cambios a los mismos para que puedan adaptarse a las nuevas necesidades. Ante situaciones así se plantea el desarrollo de nuevas aplicaciones, lo cual conlleva a una proceso de capacitación por

parte de los usuarios que tendrán que aprender a usar un nuevo sistema y adaptarse nuevamente al uso de interfaces de usuario, este proceso de cambio puede resultar un poco frustrante a los usuarios ya que en muchos casos sienten frustración al no poder realizar sus tareas con la misma rapidez como lo hacían anteriormente. Ante este panorama entonces se plantea poder diseñar la interfaz de usuario lo más parecida posible a la anterior con la modificaciones necesarias para satisfacer las necesidades de los usuarios.

Este mismo caso puede ser resuelto mediante el uso de ingeniería inversa usando el enfoque basado en modelos, debido a que con el proceso de abstracción es posible volver a generar una interfaz muy parecida a la original con las diferencias que puedan existir entre las plataformas origen y destino de las interfaz.

- Permite un soporte a la documentación de productos software

  Los modelos fueron utilizados en un inicio para transmitir ideas entre las personas involucradas en el proceso de desarrollo de una aplicación. Con la aparición del enfoque basado en modelos se han utilizado para la generación automática de interfaces de usuario (MB-UIDE). Haciendo uso de estos conceptos y de la ingeniería inversa entonces logrando la abstracción de las mismas entre los diferentes modelos podemos obtener la documentación de la interfaces de usuario.

A lo largo de este trabajo de investigación se ha venido tratando la posibilidad de poder resolver los puntos anteriormente planteados usando el poder que ofrecen los modelos y la ingeniería inversa y utilizando, con ella, los lenguajes de trasformación.

La ventaja de hacer uso del concepto de ingeniería inversa y del enfoque basado en modelos está en la posibilidad de abstraer las transformaciones y contemplar el proceso de mantenimiento de una forma, a priori, mucho más eficiente, eficaz, segura y satisfactoria.

Los trabajos realizados para la confección del capítulo tercero han permitido anticipar las posibilidades que ofrece la ingeniería inversa en los entornos de desarrollo basados en modelos para el desarrollo de interfaces de usuario. Pero la propuesta realizada en este capítulo ofrece la posibilidad de identificar diferentes actividades y tareas de investigación, que serán identificadas más adelante.

La propuesta de trabajo recogida en este capítulo puede verse integrada en diferentes actividades y proyectos de investigación que se están realizados en la actualidad y que están relacionados con la propuesta metodológica y el lenguaje UsiXML.

En los trabajos realizados hasta el momento se ha considerado la transformación del modelo de interfaz de usuario concreta a interfaz de usuario abstracta, pero ofrecer un marco metodológico que contemple todas las posibilidades de la ingeniería inversa y los entornos Mb-UID conlleva contemplar otras posibles asociaciones y transformaciones entre modelos Mb-UIDE (véase Ilustración 4.1).

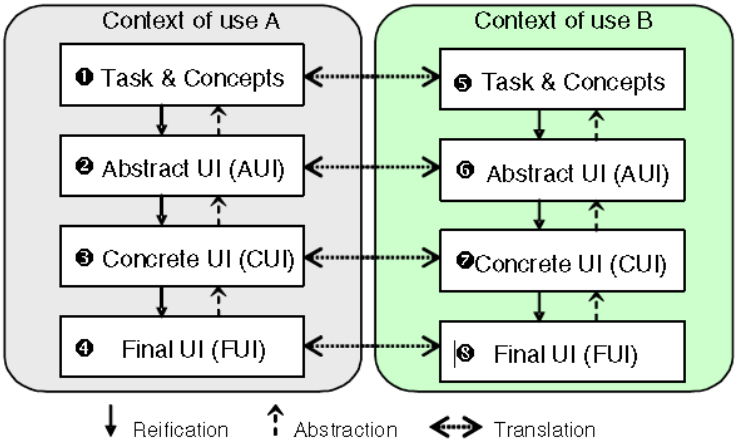

Ilustración 4-1 Marco de Trabajo UsiXML

Como parte de la propuesta de la tesis doctoral que se está planteando en este capítulo se considerara el estudio de las propuestas metodológicas que se encuentran bajo la sombría del proyecto europeo Cameleon, dentro del cual encontramos a UsiXML, TeresaXML y ConcurTaskTrees (CTT) por mencionar algunas.

Tomando en consideración lo planteado en la Ilustración 4-1 se identifica la posibilidad de poder realizar ingeniería inversa de interfaces de usuario mediante la abstracción de los modelos. El camino que se identifica en el proyecto Camelon se presenta de forma lineal, marcando una relación directa uno a uno entre cada modelo, para efectos de la propuesta de la tesis doctoral se plantea tomar como punto de partida el modelo de interfaz de usuario concreto dejando de lado la interfaz de usuario final ya que esta implica trabajar con código; por lo que se identifican 3 relaciones de abstracción entre modelos dentro del marco de trabajo del proyecto Cameleon, pero también es posible identificar relaciones que se pueden establecer entre el modelo de Interfaz de usuario concreta y el modelo de tareas y el modelo de interfaz de usuario concreta y los conceptos, estas relaciones podrían actuar como un puente para obtener los niveles de abstracción más altos definidos en Cameleon sin considerar el modelo de interfaz de usuario abstracto, estas dos últimas transformaciones serán de utilidad para aquellos usuarios que necesiten información únicamente del modelo de tareas o de los conceptos y desde ese punto poder derivar transformaciones en sentido Forward (véase Ilustración 4-2)

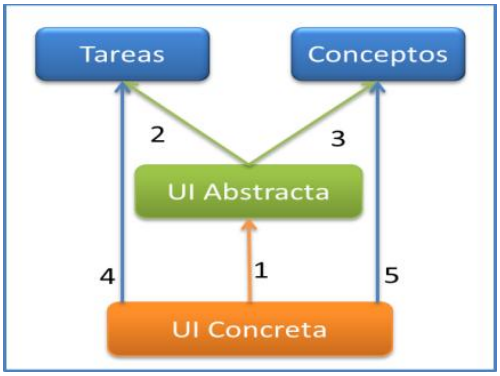

Ilustración 4-2 Abstracciones a estudiar dentro del Marco de Trabajo del Proyecto Cameleon

4.3 Metodología

A lo largo de este trabajo de investigación se ha propuesto el uso de Cameleon como metodología y marco de trabajo inicial y como lenguaje de especificación para la aplicación de técnicas de ingeniería inversa en el desarrollo dirigido por modelos, cuenta con 4 niveles de abstracción en las cuales se identifican las posibles relaciones entre los modelos, dejando marcado un camino que facilita la ingeniería inversa en el desarrollo dirigido por modelos.

Uno de los principales problemas al usar Cameleon como punto de esta, es la necesidad de seleccionar, de manera adicional y justificada, una propuesta metodológica compatible con Cameleon y que si que se encuentre descrita a un mayor nivel de concreción. En este sentido nuestra apuesta es UsiXML para el desarrollo de casos de estudio.

Se ha mencionado a lo largo de este trabajo de investigación que el marco de trabajo del proyecto Cameleon está formado por 4 niveles de abstracción (véase Ilustración 4-1), nuestra propuesta descartara el ultimo nivel de abstracción(UI Final), por lo que en esta propuesta se trabajara con 3 niveles de abstracción para las cuales se han identificado 5 posibles relaciones de abstracción entre los modelos (véase Ilustración 4-2), a continuación se describirá brevemente cada una de las relaciones de abstracción identificadas.

1. De UI Concreta a UI abstracta

   Esta transformación ya ha sido tratada en el capítulo tercero de este trabajo de investigación, dando como resultado la abstracción de un modelo de interfaz de usuario abstracta partiendo de una captura de pantalla.

   Una tareas pendiente para futuras investigaciones en esta transformación es la de transformación de ingeniería inversa de interfaces de usuario que se ejecutan en un ambiente WEB, existen muchos factores a considerar como por ejemplo cuestiones de usabilidad como la cantidad de scrolls que debe dar una usuario para la visualización completa de todo el contenido de la pagina. Otro factor a considerar en las interfaces web es que muchos de los detalles esta especificados mediante las hojas de estilo CSS por lo que es necesario determinar cómo se trataran esos elementos.

2. De UI abstracta a Tareas

   Al igual que en la transformación anterior es posible el poder identificar relaciones entre los componentes del modelo de interfaz abstracta y el modelo de tareas. Al igual que el modelo de interfaz de usuario abstracta cuenta con un número reducido de tareas, estas pueden ser de 4 tipos: tareas de usuario, tareas de aplicación, tareas de interacción y tareas abstractas, las tareas se relacionan unas con otras por medio de relaciones temporales. El problema que existe es que las relaciones temporales no se han considerado en el modelo abstracto, por lo tanto se desconocen, esto implica un problema ya que entonces esta transformación no se podría realizar, por lo que se hace indispensable buscar una forma de poder recoger esta información en la transformación del modelo de interfaz de usuario concreta al modelo de interfaz de usuario abstracta.

3. De UI Abstracta a Conceptos

Los contenedores abstractos pueden transformarse en clases y los componentes individuales de interacción transformasen en atributos o en métodos de acuerdo a la faceta que estos tengan.

4. de UI Concreta a Tareas

   Este camino es muy similar al segundo camino, se debe resolver primero el problema existente con las relaciones temporales entre las tareas. De igual forma es posible relacionar ventanas con tareas abstractas ya que estas funcionan como un contendor de tareas, dentro de una tarea abstracta se agrupan mas tareas abstracta o bien tareas de usuario, tareas de sistema o tareas de sistema.

5. de UI Concreta a Conceptos

   Partiendo desde este punto existe la posibilidad de poder obtener un diagrama de clases, donde una ventana podría representar una clase y los diferentes componentes individuales de interacción podría como etiquetas y cuadros de texto podría ser miembros de las clases, los menús y los botones de comando podrían representar métodos.

En el siguiente apartado se desglosan las actividades que se pretender realizar con el objetivo de llevar a cabo la tesis doctoral.

4.4 Tareas

Este apartado está destinado a la documentación y descripción de las principales actividades que se proponen realizar para el desarrollo de la tesis doctoral.

Actividad 1.Estado del arte

Una tarea que no debe pasar desapercibida en un trabajo de investigación es el estado del arte, aunque en este trabajo ya se ha abordado en la tarea es necesario profundizar en varios aspectos como lo son:

Tarea 1. Estudio de la propuesta CAMELEON y de la Metodología UsiXML y otras propuestas relacionadas con la especificación de interfaces de usuario.

Tarea 2. Estudio de lenguajes de especificación de transformaciones (realizado en el trabajo de investigación incluido en el capítulo tercero de este documento).

Tarea 3. Estudio de las propuestas relacionadas con ingeniería inversa (reverse engineering) tanto en general, como aquellas asociadas al desarrollo de interfaces de usuario.

Actividad 2.Definición de la propuesta

En esta sección se procederá a la definición de la propuesta, donde se han podido identificar algunas de las siguientes tareas:

Tarea 1. Definición del alcance y dimensión de la metodología de desarrollo de interfaces de usuario considerando el sentido inverso, es decir, aquel en el que se parte de especificaciones concretas y se alcanzan modelos independientes de la plataforma.

Tarea 2. Identificación de los diferentes caminos que se pueden seguir en el proceso de ingeniería inversa en el desarrollo dirigido por modelos (realizado parcialmente en el trabajo de investigación recopilado en el capítulo tercero de este documento).

Tarea 3. Identificación de relaciones o influencias que puedan surgir entre los diferentes caminos identificados, es decir, entre los distintos modelos que aparecen considerados en las propuestas Mb-UIDE.

Tarea 4. Estudio de herramientas susceptibles de ser utilizadas para dar soporte al proceso inverso en el desarrollo de interfaces de usuario.

Actividad 3. Desarrollo de la propuesta

Tarea 1. Definición transformaciones entre los modelos considerados en el proceso de desarrollo de interfaces de usuario inverso.

Tarea 2. Desarrollo de herramientas de apoyo a la especificación de los modelos y transformaciones identificados en la especificación de interfaces de usuario.

Tarea 3. Implementación de las reglas de transformación de modelos haciendo uso de lenguajes de transformación.

Tarea 4. Incorporación de herramientas al marco de trabajo metodológico propuesto.

Actividad 4. Experimentación y validación

Tarea 1. Evaluación, atendiendo a diferentes criterios de calidad, del marco de trabajo propuesto para abordar la generación inversa de interfaces de usuario.

Tarea 2. Identificación y utilización de distintos casos de estudio para verificar y validar los desarrollos y marco de transformaciones realizado.

Actividad 5. Documentación, diseminación y difusión

El tiempo dedicado a estas actividades se describe en Ilustración 4-3**¡Error! No se encuentra el origen de la referencia.** del siguiente apartado.

4.5 Cronograma

Este apartado describe la planificación temporal que se da a cada una de las actividades y tareas que se realizaran en el desarrollo de la tesis doctoral. La planificación esta programada para que se desarrolle en un tiempo aproximado de 3 años, algunas de las tareas que se plantean ya se han comenzado a tratar para efectos de este trabajo de investigación.

Se propone que el estudio del arte sea revisado constantemente durante el periodo que dure la investigación que culminara con la redacción de la tesis doctoral, se hace necesario verificar aquellas propuestas relacionadas con la ingeniería Inversa de interfaces de usuario que surjan a lo largo del periodo de investigación.

La Ilustración 4-3 muestra la planificación del trabajo de investigación, está dividido en 5 actividades principales que a su vez se subdividen en varias tareas que se deben seguir para completar las mismas.

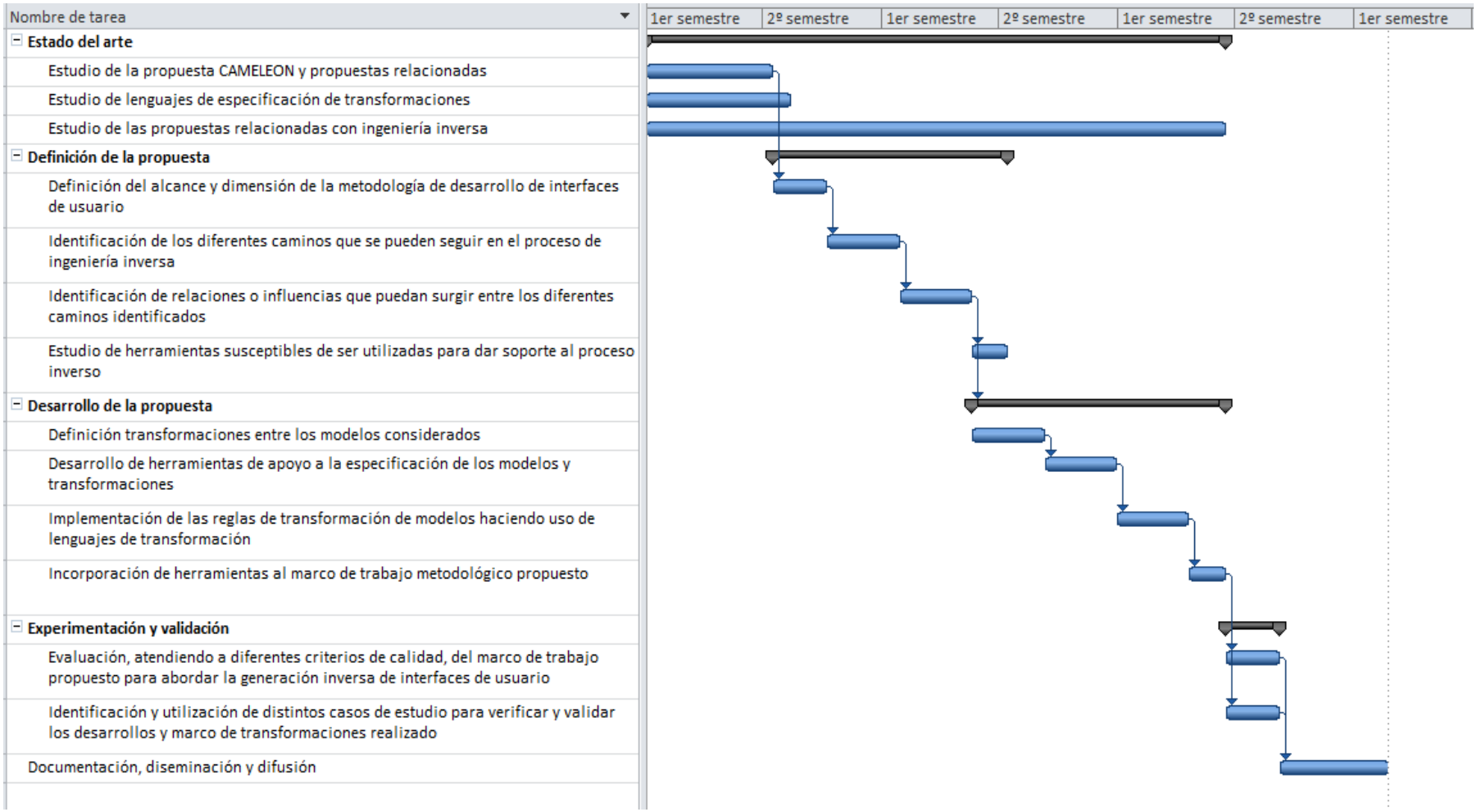

Ilustración 4-3. Cronograma de actividades y tareas

Un punto importante a considerar dentro de esta planificación es la elección de la propuesta o las propuestas metodológicas que se usarán para lograr la abstracción de los modelos. Sin definir este punto no se podría realizar nuestra propuesta y la identificación de los posibles caminos a seguir sería imposible de establecer. Por ello se hace necesario que esta tarea se encuentre finalizada para definir la propuesta. Una vez definida la metodología será posible identificar los caminos que se pueden dar entre los diferentes modelos y las posibles influencias que puedan ejercer las relaciones entre modelos.

El estudio de herramientas que puedan dar soporte al proceso inverso viene enfocado a la posibilidad de modelar esas interfaces permitiendo la obtención de los ficheros que contendrán la especificación de las interfaces gráficas en formato XML, esta tarea puede llegar a realizarse varias veces, dado que es posible que surjan nuevas herramientas durante el proceso que dure la investigación. De propiciarse el surgimiento de nuevas herramientas sería interesante el estudio de las mismas con el objetivo de ver de qué forma pueden ayudar en el proceso de ingeniería inversa. Tomando en cuenta que lo que se pretende es partir de conjuntos de capturas de pantallas a las que se quiere aplicar ingeniería inversa, es posible que sea necesaria el desarrollo de nuevas herramientas para la especificación de las interfaces de usuario, ya que para lograr relaciones de abstracción se necesita una especificación en formato XML que responda a las características de una metodología (como sucedía en el caso de estudio recogido en el capítulo tercero).

Una vez definidas la propuesta y la metodología será posible establecer las relaciones entre los modelos y con el uso de un lenguaje de transformación que ayuda en la realización de estas tareas se podrán definir las reglas que transformaran entre modelos. Es posible que una vez resuelto este punto sea necesario el desarrollo de herramientas que automaticen el proceso de ingeniería inversa, por lo que se considera en la planificación una tarea que aborda esta posibilidad.

La experimentación será llevada a cabo mediante casos de estudio en los que se buscaran relaciones de abstracción entre los diferentes modelos. Así como atendiendo a aquellos criterios de calidad relacionados con el tema objeto de investigación.

Una vez que se han hayan realizado casos de estudio será posible la documentación de la misma, así como la publicación en diferentes foros, tales como revistas y congresos de reconocido prestigio, de los que se podrá obtener retroalimentación.

Capítulo 5

Currículum Vitae

5.1 Información Personal

Nombre: Reynaldo José Cruz Ocampo

Pasaporte #: Z045572

Nacionalidad: Hondureña

Dirección: Calle Hellín # 60, Residencia José Isbert

Albacete, 02002, Albacete, España

Correo: reynaldojose.cruz@alu.uclm.es

Teléfono: 655 60 97 42

5.2 Titulación

Ingeniero En Ciencias de la Computación, Universidad Católica de Honduras

Noviembre 2009

Nota Media: 84/100

5.3 Becas

Titulo: Fundación Carolina

Objetivo: Actividades de Investigación

Organismo: Fundación Carolina

5.4 Cursos

Titulo: Gestión de Calidad Total

Lugar: La Ceiba, Atlántida, Honduras

Organizador: Universidad Católica de Honduras

Titulo: Cisco CCNA -1

Lugar: Academia Local CISCO Universidad Católica de Honduras

Organizador: Universidad católica de Honduras

Titulo: Diplomado de Ingles

Lugar: La Ceiba, Atlántida, Honduras

Organizador: Universidad Católica de Honduras

5.5 Intereses especiales

Desarrollo de Interfaces de Usuario Dirigido por Modelos

Desarrollo de Software

Administración de Bases de Datos

5.6 Experiencia Laboral

Empresa: Standard Fruit de Honduras

Departamento: Tecnologías de la Información

Actividades: Desarrollo de Módulos para el Sistemas de Información Bananero.

Empresa: ENERGUA, S.A

Departamento: Sistemas

Actividades: Mantenimiento Preventivo y Correctivo de Equipo, Administración de Bases de Datos, Respaldos de Servidores, Administración y Mantenimiento de Redes, Análisis y Desarrollo de Aplicaciones, Soporte en Escritorio a Usuarios.

Empresa: FreeLancer

Actividades: Desarrollo de Soluciones a la Medida, Generalmente Control de Inventarios y Puntos de Venta.

Bibliografía

Abrams, M., Phanouriou, C., Batongbacal, A. L., Williams, S. M., & Shuster, J. E. (n.d.). Retrieved 03 01, 2012, from http://www8.org/w8-papers/5b-hypertext-media/uiml/uiml.html

AGG Homepage. (n.d.). Retrieved 2012, from http://user.cs.tu-berlin.de/~gragra/agg/

ATLAS group. (2005). *ATL: Atlas Transformation Language. ATL Starter's Guide.*

Banerjee, I., & Nagarajan, A. (2003). GUI Ripping: Reverse Engineering of Graphical User Interfaces for Testing. *10th Working Conference on Reverse Engineering 2003 WCRE 2003 Proceedings (2003)* (pp. 260-269). Department of Computer Science, University of Maryland: IEEE.

Belaunde, M., & Dupe, G. (n.d.). *SmartQVT Home Page.* Retrieved 06 12, 2012, from http://smartqvt.elibel.tm.fr/index.html

Berti, S., Correani, F., Paterno, F., & Santoro, C. (2004). The TERESA XML Language for the Description of Interactive Systems at Multiple Abstraction Levels. *Proceedings of the Workshop on Developing User Interfaces with XML Advances on User Interface Description Languages (2004)*, (pp. 103-110). Pisa.

Borland Software Corporation. (2007). *Borland.* Retrieved 06 17, 2012, from http://techpubs.borland.com/together/2007/EN/TogetherDSLToolkit.pdf

Bouillon, L. (2006). *Reverse Engineering of Declarative User Interfaces.* Louvain: Phd Thesis, Université catholique de Louvain.

Bouillon, L., Limbourg, Q., Vanderdonckt, J., & Michotte, B. (2005). Reverse Engineering of Web Pages based on Derivations and Transformations. *Web Congress, 2005. LA-WEB 2005. Third Latin American.* IEEE Computer Society.

Chikosfsky, E., & Cross, J. (1990). Reverse Engineering and Design Recovery: A Taxonomy. *IEEE Software (1990) , 7* (1), 13-17.

Coimbra Morgado, I., & Pascoal Faria, J. (2011). Reverse Engineering of Graphical User Interfaces. *The Sixth International Conference on Software Engineering Advances* (pp. 293-298). Barcelona: IARIA.

Eclipse. (n.d.). Retrieved 04 29, 2012, from http://www.eclipse.org: http://www.eclipse.org/atl/

ECLIPSE. (n.d.). *ECLIPSE*. Retrieved 05 17, 2012, from ELICPSE/ATL: http://www.eclipse.org/atl/

Ehrig, H., Herig, K., & Prange, U. (2006). *Fundamentals of Algebraic Grahp Transformation.* Berlin: Springer.

Eilam, E. (2005). *Reversing: Secrets of Reverse Engineering.* Wiley Publishing, Inc.

European Commission Information Society Technologies. (n.d.). *Cameleon Project*. Retrieved 05 20, 2012, from http://giove.isti.cnr.it/projects/cameleon.html

Favre, L. (2010). *Model Driven Architecture for Reverse Engineering Technologies, Strategic Directions and System Evolutions.* New York: Hershey.

Foley, J., Kim, W., Kovacevic, S., & Murray, K. (1986). The User Interface Design Environment - A Computer Aided Software Engineering Tool for the User Computer Interface. *IEEE Software 6* , 1,25-36.

Gardner, T., & Griffin, C. (2003). A review of OMG MOF 2.0 Query / Views / Transformations Submissions and Recommendations towards the final Standard. In *MetaModelling for MDA (2003)* (pp. 293-298). Citeseer.

Giandini, R. S. (2007). *Un Marco Formal para Transformaciones en la Ingeniería de Software Conducida por Modelos.* Buenos Aires, Argentina: PHD Thesis, Universidad Nacional de La Plata.

Harold , E. R., & Means, W. S. (2005). XML Imprescindible. Anaya.

Hex-Rays. (2012, 02 12). *HEX-RAYS*. Retrieved 02 12, 2012, from http://www.hex-rays.com/products/ida/support/index.shtml

HIIS Laboratory. (n.d.). *HIIS Laboratory The Human-Computer Interaction Group*. Retrieved 03 06, 2012, from http://giove.isti.cnr.it/ctte.html

IBM. (n.d.). *IBM Rational*. Retrieved 06 01, 2012, from http://www-01.ibm.com/software/rational/

Jardim Nunes, N., & Falcão e Cunha, J. (2000). Towards a UML Profile for Interaction Design: The Wisdom Approach. *Lecture Notes in Computer Science (2000) , 1939*, 101-116.

JDec. (n.d.). *Jdec*. Retrieved 02 12, 2012, from http://jdec.sourceforge.net/

Jouault, F., & Kurtev, I. (2005). Transforming Models with ATL. *J.-M. Bruel (Ed.): MoDELS 2005 Workshops, LNCS 3844* , pp. 128 – 138.

Jouault, F., Allilaire, F., Bézivin, J., & Kurtev, I. (2008). ATL: A model transformation tool. *Science of Computer Programming 72* , 31-39.

Kleppe, A., Warmer, J., & Bast, W. (2003). MDA Explained: The Model Driven Architecture: Practice and Promise. *AddisonWesley Professional , 83* (7), 192.

Kollmann, R., & Gogolla, M. (2001). Application of UML Associations and Their Adornments in Design Recovery. (P. Aiken, & E. Burd, Eds.) *8th Working Conference on Reverse Engineering* , 89-90.

Kollmann, R., Selonen, P., Stroulia, E., Systa, T., & Zundorf, A. (2002). A Study on the Current State of the Art in Tool-Supported UML-Based Static Reverse Engineering. *Ninth Working Conference on Reverse Engineering 2002 Proceedings (2002)* (pp. 22-32). IEEE Comput. Soc.

Lara, J., & Vangheluwe, H. (2002). AToM3: A Tool for Multi-Formalism Modelling and Meta-Modelling. *LNCS 2306* , 174 - 188.

Levendovszky, T., Lengyel, L., Mezei, G., & Charaf, H. (2004). A Systematic Approach to Metamodeling Environments and Model Transformation Systems in VMTS. *Electronic Notes in Theoretical Computer Science (2004) , 127* (1), 65-75.

Limbourg, Q. (2004). *Multi-Path Development of User Interfaces.* Louvain, Belgica: PHD Thesis,Université catholique de Louvain.

Limbourg, Q., & Vanderdonckt, J. (2004). Transformational Development of User Interface with Graph Transformations. In *Computer-Aided Design of User Interfaces IV* (pp. 107–120). Kluwer Academic Publishers.

Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., & López-Jaquero, V. (2005). USIXML: A Language Supporting Multi-path Development of User Interfaces. *LNCS* , 200-220.

Lopez Jaquero, V. (2005). *Interfaces de Usuario Adaptativas Basadas en Modelos y Agentes de Software.* Albacete, España: PHD Thesis.

Lozano, M. D., Gonzalez, P., & Ramos, I. (2001). *Entorno metodológico orientado a objetos para la especificación y desarrollo de interfaces de usuario.* Albacete, Universidad Castilla - La Mancha: Tesis Doctoral.

Merlo, E., Gagné, P. Y., & Thibout, T. A. (1994). Inference of Graphical AUIDL Specifications for the Reverse Engineering of User Interfaces. *Proceedings of the International Conference on Software Maintenance ICSM94 Victoria BC Sept 1923 1994 (1994)* (pp. 80-88). Victoria.

Michotte, B., & Vanderdonckt, J. (2008). GrafiXML, A Multi-Target User Interface Builder based on UsiXML. *Fourth International Conference on Autonomic and Autonomous Systems ICAS08 (2008)* (pp. 15-22). IEEE.

Michotte, B., & Vanderdonckt, J. (2008). *GrafiXML, A Multi-Target User Interface Builder based on UsiXML.* IEEE Computer Society.

Microsoft. (2012, 02 12). *Microsoft MSDN*. Retrieved 02 12, 2012, from http://msdn.microsoft.com/es-es/library/f7dy01k1(v=vs.80).aspx

Montero, F., & Lopez-Jaquero, V. IDEALXML: AN INTERACTION DESIGN TOOL A Task-Based Approach to User Interfaces Design. *ComputerAided Design of User Interfaces V* (pp. 245-252). Springer Netherlands.

Mozilla. (n.d.). *Mozilla*. Retrieved 03 02, 2011, from https://developer.mozilla.org/es/XUL

Mozilla. (n.d.). *Mozilla*. Retrieved 04 15, 2012, from https://developer.mozilla.org/es/XSLT

MUHAIRAT, I. M., AL-QUTAISH, R. E., & ATHAMENA, B. M. (2011). FROM GRAPHICAL USER INTERFACE TO DOMAIN CLASS DIAGRAM: A REVERSE ENGINEERING APPROACH. *Journal of Theoretical and Applied Information Technology , 24* (1), 28-40.

OMG. (2003). *MDA Guide Version 1.0.1.*

OMG. (2005). MOF QVT Final Adopted Specification. Object Management Group.

OMG. (2006). *Object Constraint Language, OMG Available Specification,.*

OMG. (2003). *OMG Meta Object Facility (MOF) Core Specification.*

Paternò, F. (1997). ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. *Interface (1997) , 96*, 362-369.

Pinheiro da Silva, P., & W. Paton, N. (2003). User Interfaces Modeling in UMLi. *IEEE Software (2003) , 20* (4), 62-69.

Pinheiro, P. (2002). *Object Modelling of Interactive Systems: The UMLi Approach.* Manchester, Inglaterra: Phd Thesis, University of Manchester.

Puerta, A. (1997). A Model-Based Interface Development Environment. *IEEE Software* , 40-47.

Ratiu, D. (2009). Reverse Engineering Domain Models from Source Code. *Technische Universit¨at M¨unchen* , 1-4.

Rusty Harold, E., & Scott Means, W. (2005). *XML Imprescindible.* ANAYA.

Sánchez Ramón, Ó., Sánchez Cuadrado, J., & García Molina, J. (2010, September). Model-Driven Reverse Engineering of Legacy Graphical User Interfaces. *ASE'10,* , 20–24.

Schlungbaum, E. (1996). Model-based User Interface Software Tools, Current state of declarative models. *Technology (1996)* .

Stroulia, E., El-Ramly, M., Kong, L., & Sorenson, P. (1999). Reverse Engineering Legacy Interfaces: An Interaction-Driven Approach. *WCRE 1999 Proceedings of the 6th Working Conference on Reverse Engineering (1999)* (pp. 292-302). IEEE.

Szekely, P., Sukaviriya, P., Castells, P., Muthukumarasamy, J., & Salcher, E. (1995). Declarative Interface Models for User Interface Construction Tools: the MASTERMIND Approach. In *Information Sciences (1995)* (pp. 120-150). Chapman & Hall.

Taentzer, G., Ehrig, K., Guerra, E., de Lara, J., Lengyel, L., Levendovszky, T., et al. (2005). Model Transformation by Graph Transformation: A Comparative Study. *Technology (2005)* , 195.

Technologies, E. C. (n.d.). *Cameleon Project Plasticity of user interfaces*. Retrieved 03 16, 2012, from http://giove.isti.cnr.it/projects/cameleon.html

The World Wide Web Consortium. (n.d.). *W3C*. Retrieved 06 03, 2012, from http://www.w3.org/2008/10/mbui/W3C%20Incubator-paterno.pdf

USIXML. (n.d.). *USIXML*. Retrieved 03 03, 2012, from http://www.usixml.org

Varro, D., & Pataricza, A. (2004). Generic and meta-transformations for model transfortransformation. *Proc. UML 2004: 7th International Conference on the Unified Modeling Language*, (pp. 290–304). Lisboa, Portugal.

W3C. (n.d.). *W3C*. Retrieved 04 15, 2012, from http://www.w3.org/standards/xml/transformation#summary

W3C. (n.d.). *W3C.* Retrieved 06 17, 2012, from Extensible Markup Language: http://www.w3.org/TR/2008/REC-xml-20081126/

W3C. (1999). *XSL Transformations (XSLT).*

WANG, W. (2011). *REVERSE ENGINEERING: TECHNOLOGY OF REINVENTION.* CRC Press.